Math

UIUCDCS-R-77-859

UILU-ENG 77 1712

SCHEDULING PERIODIC-TIME-CRITICAL JOBS ON SINGLE PROCESSOR AND MULTIPROCESSOR COMPUTING SYSTEMS

by

SUDARSHAN KUMAR DHALL

April, 1977

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

SCHEDULING PERIODIC-TIME-CRITICAL JOBS ON SINGLE PROCESSOR AND MULTIPROCESSOR COMPUTING SYSTEMS

BY

SUDARSHAN KUMAR DHALL

B.A., Panjab University, 1956
M.A., University of Delhi, 1968
M.S., University of Illinois, 1972

THESIS

Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1977

Urbana, Illinois

* This work was supported in part by the National Science Foundation under Grant Number NCS 73-03408.

ACKNOWLEDGMENT

I wish to express my sincere thanks and gratitude to my thesis advisor, Professor Chung-Laung Liu, whose supervision, invaluable guidance and advice, and constant encouragement made this work possible. I would also like to thank Professor Paul Handler for supporting me during the course of the study.

Lastly, I wish to thank my wife, Pushpa, for her patience, encouragement and devoted love.

This work is dedicated to my parents.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Motivation and Objectives

There is a variety of situations in which a set of resources is available for the performance of a set of jobs. The problem of scheduling is to allocate the resources to the jobs so that they will be performed in accordance with some given constraints. We mention here some examples:

(a) In a diagnostic clinic, a patient is scheduled to undergo a number of tests and to visit a number of specialists. In many cases, the schedule must ensure that a patient has undergone certain tests before his visit to a certain specialist.

(b) A number of input-output devices are connected to a computer through a multiplex channel. It is desired to allow these devices to access the central memory through the multiplex channel.

(c) At an airport with a number of runways, it is necessary to assign aircrafts to runways for landing or taking off in a certain order.

(d) The manager of a service station with three mechanics wants to ensure servicing ten vehicles during the day. He has to decide which mechanic should work on which vehicle at what time. In assigning work to mechanics, he may also have to take into consideration the capability of the individual mechanic with regard to the type of work he is being assigned. It may at times be necessary to ask a mechanic to discontinue work on one vehicle to start work on another.

In the above examples, "specialists", "central memory of the computer", the "runways" and the "mechanics" are resources and "visits of outpatients", "accessing by the input-output devices", "take off or landing of an aircraft", and "servicing the vehicles" are jobs to be performed. Regardless of the type of resources available and the character of the jobs to be performed, there is a fundamental similarity among all these scheduling problems: given a set of jobs and a set of resources, to determine the allocation of resources to jobs and the order in which the jobs will be executed on the allocated resources so as to meet some desired goal. While determining such an allocation and the ordering one should satisfy the given properties of, and constraints on, the jobs and resources. For example, for a given set of jobs it may not be possible to start execution of a particular job before certain other job or jobs have been completed, as in an assembly line where a certain number of parts must be ready before a unit can be assembled. It may also be the case that a particular job in the set cannot be executed on a particular resource. With each scheduling problem, one may associate a cost function. For example, if some machinery is hired for executing a set of jobs, it will be advantageous if all the jobs can be completed as soon as possible. Thus, in many real life problems, poor scheduling decisions might lead to excessively large costs. In certain other cases, it is necessary to complete jobs before some prescribed deadlines, or else it may result in some irreparable loss. It would, therefore, be worth considering those cases where a proper scheduling decision may result in savings, or, in general, minimize a cost function. A special case is where the cost function is a function

of the deadline. Considerable effort has been spent in research in this direction during the past years. Many interesting results have been obtained and some intriguing questions are still unanswered. This thesis concerns itself with one aspect of the general scheduling problem - that of scheduling a set of jobs with a view to meeting the deadlines of all jobs.

1.2 The General Problem

The general scheduling problem is to allocate available resources over a period of time to perform a set of jobs so as to achieve some performance measure. We shall use the language of computing, and henceforth a resource will be referred to as a processor. A set of processors will be referred to as a computing system. Thus we may talk of a single-processor computing system or a multi-processor computing system. The processors in a computing system might not be identical. It may be that it takes different units of time to execute a job on different processors. It may also be possible that some jobs can only be executed on some of the processors. A computing system with n processors will be denoted by $\mathcal{P} = \{P_1, P_2, \ldots\ldots, P_n\}$.

A set of jobs is formally specified by an ordered triple $(\mathcal{J}, \prec, \mu)$. $\mathcal{J} = \{J_1, J_2, \ldots\ldots, J_m\}$ denotes a set of m jobs to be executed. $\prec$ is a partial ordering relation defined on $\mathcal{J}$, which specifies restrictions on the order in which jobs can be performed. That is $J_r \prec J_s$ means that the execution of job J_s cannot begin until the execution of job J_r has been completed. J_r is called a predecessor of J_s and J_s is called a successor of J_r. J_s is said to be an immediate successor of J_r, if there is no J_t such that $J_r \prec J_t \prec J_s$.

A set of jobs is said to be independent if the partial ordering relation $\prec$ is empty.

μ is a function from $\mathcal{J}$ to the space of n-component vectors. $\mu(J_i) = (t_{1i}, t_{2i}, \ldots\ldots, t_{ni})$, where $0 \leq t_{ri} \leq \infty$, and for each i there exists at least one r such that $t_{ri} < \infty$. The value of t_{ri} is the time it takes to execute job J_i on the r^{th} processor and is called the execution time of job J_i on that processor. That $t_{ri} = \infty$ means that job J_i cannot be executed on the r^{th} processor. If all processors in the computing system are identical, then t_{ri} is the same for all r. In that case, we will simply write $\mu(J_i) = t_i$.

A set of jobs can be represented by a directed graph such as the one in figure 1.1. Corresponding to each job J_i in the set there is a node J_i in the graph. There is a directed edge from the node J_r to the node J_s if and only if J_s is an immediate successor of J_r. The node corresponding to job J_i is labelled as $J_i/\mu(J_i)$.

By scheduling a set of jobs on a multiprocessor system we mean to specify for each job J_i the time interval(s) within which it will be executed and the processor P_r on which execution will take place. Such a specification of the assignment of processors to jobs is called a schedule. An explicit way to describe a schedule is a timing diagram, which is also known as the Gantt Chart. As an example, the timing diagram of a schedule for execution of the set of jobs in figure 1.1 on a three-processor computing system is shown in figure 1.2.

In a schedule a processor might be left idle either because there is no executable job at that time or because it is an intentional choice. (A job is said to be executable at a certain time instant if

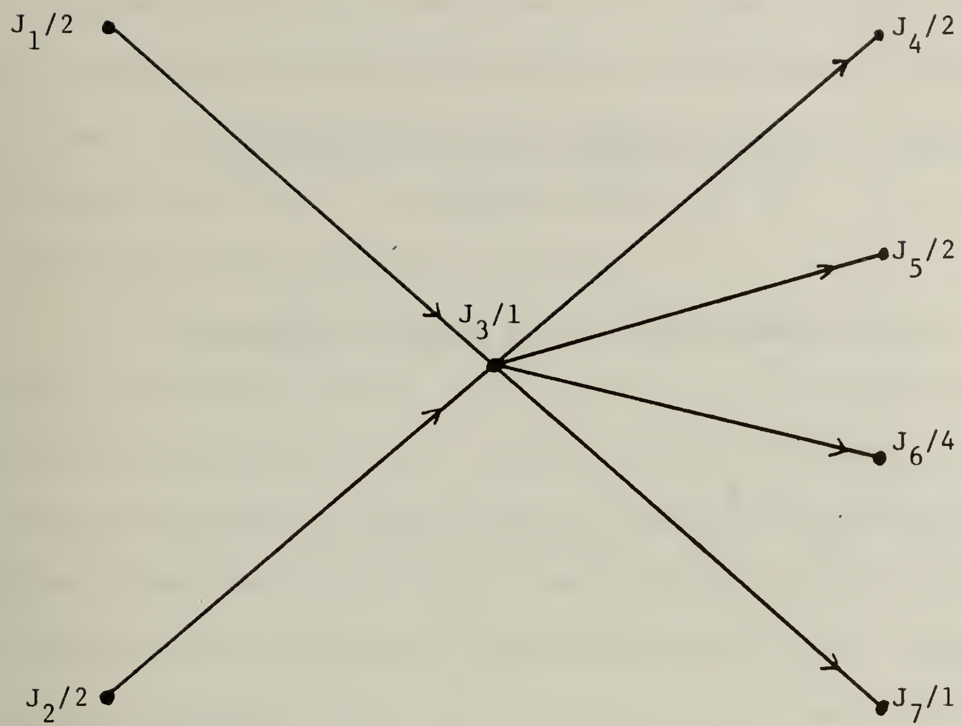

Figure 1.1 A directed graph representing a set of jobs.

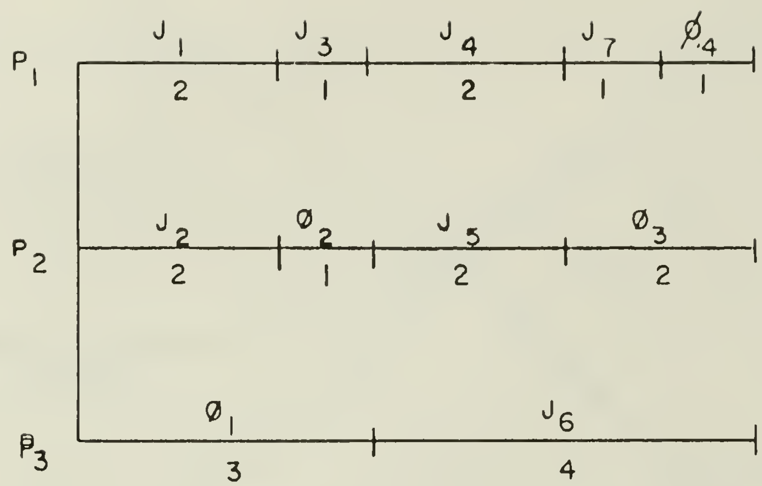

Figure 1.2 A timing diagram representing a schedule

the execution of all its predecessors has been completed at that time.) Clearly, it is neither necessary nor beneficial in a schedule to leave all processors idle at the same time. For a given schedule, an <u>idle period</u> of a processor is defined to be the time interval during which a processor is not executing any job (while at least one other processor is executing some job). In the Gantt Chart, ϕ_1, ϕ_2, ... are used to denote such idle periods (figure 1.2).

A <u>scheduling algorithm</u> is a procedure that produces a schedule for every given set of jobs. A class of scheduling algorithms may have the restriction that a processor will not be left idle intentionally. That is, a processor is left idle for a certain period of time if and only if there is no executable job within that period. A scheduling algorithm that follows this rule is called a <u>demand scheduling algorithm</u>. Such a scheduling algorithm can be specified by merely giving the rules on how jobs are to be chosen for execution at any instant when one ore more processor is free. (Of course, the choice is only among jobs that are executable at that time.) A subclass of demand scheduling algorithms is priority driven scheduling algorithms. According to a <u>priority driven scheduling algorithm</u> all jobs in the set are assigned priorities and jobs with higher priorities are executed instead of jobs with lower priorities when they are competing for processors. Rules are also provided for breaking ties. In this type of algorithm, one just lists the jobs in $\mathcal{J}$ in descending order of their priorities from left to right. Such a list is called a <u>priority list</u>. When a processor becomes free, the priority list is scanned from left to right until the first executable job, which has not yet

been executed, is found. This job is then assigned to the free processor. (If two or more processors are available at the same time, a rule is also specified as to which of the processors will be assigned which job. A simple rule is to assign a higher priority job to a processor of lower index.) Such a scheduling algorithm is also sometimes referred to as a list scheduling algorithm.

A scheduling algorithm is said to be non-preemptive if it follows the rule that once the execution of a job has begun, it must continue without interruption until completion. On the other hand, a preemptive scheduling algorithm is the one that permits the execution of a job to be interrupted and removed from the processor subject to the condition that the interrupted job is restarted later from the point at which it was last interrupted. A possible interpretation of the preemptive strategy is: suppose a high speed drum is available in a multiprocessor computing system. It is then quite reasonable to execute a portion of a program, store all intermediate results on the high speed drum to make room (on the processor) for another program and retrieve the results when execution of this program is resumed later on.

The completion time, $\omega(S)$, of a given schedule S, is the total time it takes to complete the execution of all jobs according to the schedule S. Assuming that the starting time of a schedule is zero, let $f_i(S)$ denote the completion time of job J_i according to schedule S. Then the completion time can be represented symbolically as:

$$\omega(S) = \max_{1 \leq i \leq m} f_i(S),$$

where m is the number of jobs in the set.

The <u>mean</u> <u>completion</u> (<u>or</u> <u>flow</u>) <u>time</u> of a set of m jobs according to a schedule S is defined as:

$$\tilde{\omega}(S) = \frac{1}{m} \sum_{i=1}^{m} f_i(S).$$

Different criteria can be used to measure the performance of a schedule. Some of the common measures of performance are the completion time and the mean flow time. A schedule S is said to be <u>optimal</u> with respect to a certain measure of performance if it minimizes the chosen performance index. Thus if completion time is used as the measure of performance for a schedule then clearly for a given set of jobs, a 'good' schedule is one which has 'short' completion time and an optimal schedule is one which has shortest possible completion time.

As mentioned earlier a scheduling algorithm is a procedure that produces a schedule for every scheduling problem in a class for which it is defined. A <u>scheduling</u> <u>algorithm</u> is <u>optimal</u> with respect to a measure of performance if it produces an optimal schedule for every scheduling problem in a class of problems. In case a scheduling algorithm is not optimal, one would naturally like to determine its effectiveness. The effectiveness of a scheduling algorithm is measured by how 'good' the schedules it produces are. Two of the most common measures for evaluating the performance of an algorithm are its average performance and its worst-case performance. To study the average performance of algorithms is known as a <u>statistical</u> <u>analysis</u> and to study the worst-case performance of algorithms is known as a <u>combinatorial</u> <u>analysis</u> of algorithms. For obtaining results about

average performance, certain assumptions concerning the statistical distribution of the parameters involved are required. Since an algorithm works on a large variety of input data, such a statistical distribution may not be reliably obtainable. Most of the current work is concerned with the combinatorial analysis of scheduling algorithms. In addition to the fact that the worst-case behavior does bound the average behavior, there are many situations in which worst-case performance is an appropriate measure. For example, before implementing an algorithm that determines how traffic should be directed to maximize the total traffic flow, it would be desirable to study its worst-case performance rather than its average performance. It will be the worst-case behavior that will be evaluated for the algorithms presented in this thesis.

1.3 A Survey of Previous Work

The problem of scheduling a set of jobs ($\mathcal{J}$, $\prec$, μ) on an n-processor computing system to minimize the completion time has been studied extensively. Unfortunately, very little is known about 'efficient' algorithms that produce optimal schedules for arbitrary sets of jobs. An algorithm is considered to be efficient if the number of elementary steps of the algorithm is bounded by a polynomial function of the size of the problem, which in this case is the number of jobs to be scheduled. Such algorithms are usually referred to as polynomial-time-bounded algorithms, or simply polynomial algorithms. Since for a set of m jobs there is only a finite number of schedules, one could find an optimal schedule by examining all such schedules. Finding an optimal schedule by such exhaustive search is clearly inefficient,

since the number of elementary steps involved is exponential in m. It is at least partially comforting to know that the general scheduling problem falls in the category of what is known as non-deterministic polynomial-time-hard (NP hard) problems. A problem is said to be NP-hard if given a deterministic polynomial time algorithm for the problem, it is possible to use that algorithm to obtain a deterministic polynomial time algorithm for every problem for which there exists a non-deterministic polynomial-time algorithm. Roughly speaking, a problem is said to be NP-complete if it is NP-hard and if there exists a non-deterministic polynomial-time algorithm for it. There are many classical problems in combinatorics such as the travelling salesman problem, the problem of determining a maximum clique of an undirected graph, the question whether a logical expression is satisfiable, and the 0-1 integer programming problem, for which no deterministic polynomial-time algorithm which produces an optimal solution is known. The first two of these problems are known to be NP-hard and the last two are known to be NP-complete (see Cook [3] and Karp [9]). Many of these problems are of practical importance and a great deal of time and effort has been spent on them to find a polynomial-time-bounded algorithm. Since so far no polynomial-time-bounded algorithm has been found for even one of them, it is not unreasonable to conjecture that no such polynomial-time algorithm exists. However, in spite of the overwhelming empirical evidence to the contrary, it is still an open question whether NP-complete problems admit of a polynomial-time-bounded solution.

As mentioned earlier, it has been shown that the general scheduling problem is an NP-hard problem. Furthermore, the following

simplified versions of the general scheduling problem have been shown to be NP-complete (see Ullman [17]):

(1) All jobs in the given set of jobs have equal execution time and the number of processors in the computing system is arbitrary. (Processors are assumed to be identical in all respects.)

(2) The execution time of each job in the set is either one or two and there are only two identical processors in the computing system.

Several authors have designed efficient algorithms to produce optimal schedules (see Hu [7], Fujii, Kasami, & Ninomiya [4], and Coffman & Graham [2]). Unfortunately, these algorithms are applicable to some special cases only. As a matter of fact, efficient algorithms that produce optimal schedules are known only for the following two special cases:

(1) Jobs having unit execution times with the precedence relation over them being a forest are to be scheduled on a computing system with identical processors (see Hu [7]).

(2) Jobs having unit execution times are to be executed on a computing system with two identical processors (see Fujii, Kasami, & Ninomiya [4], and Coffman and Graham [2]).

In view of the difficulty in designing an optimal scheduling algorithm for an arbitrary set of jobs one might wish to consider the approach of designing algorithms which do not require much effort to produce a 'reasonably good' schedule. Thus it may be interesting to consider algorithms which are easy to implement, although they may not necessarily yield an optimal schedule. One might consider a nearly-optimal schedule quite satisfactory, if it is easier to implement

such a schedule. A classical result in this direction is due to Graham [6] which may be stated as follows:

Theorem: For a computing system with n identical processors, let ω denote the completion time of a schedule for a given set of jobs when scheduled according to an arbitrary priority list and let ω_0 denote the shortest possible completion time. Then

$$\frac{\omega}{\omega_0} \leqq 2 - \frac{1}{n} .$$

Moreover, the bound is best possible in the sense that the right hand side of the above equation cannot be replaced by any smaller function of n.

The significance of this result is that in terms of the completion time a schedule produced by any list scheduling algorithm is not worse than an optimal schedule by more than 100 %.

In the case of computing systems with processors of different speeds, Liu & Liu [14] obtained the following result:

Theorem: For a computing system with n_1 processors of speed b_1, n_2 processors of speed b_2, ... , n_k processors with speed b_k, where $b_1 > b_2 > \ldots\ldots > b_k \geqq 1$, let ω denote the completion time when jobs are scheduled according to an arbitrary priority list, and let ω_0 denote the shortest possible completion time. Then

$$\frac{\omega}{\omega_0} \leqq 1 + \frac{b_1}{b_k} - \frac{b_1}{\sum_{i=1}^{k} n_i b_i} .$$

Recently Garey and Graham [5] considered an augmented multiprocessing model. They introduced a set of "auxiliary resources" into the system with the restriction that at no time may the system

use more than some predetermined amount of each auxiliary resource. This can be described as follows.

Assume that a set of auxiliary resources $\mathcal{R} = \{R_1, R_2, .. , R_s\}$ is given and that these resources have the following properties. The total amount of resource R_i available at any time is (normalized without loss of generality to) 1. For each k, the job J_k requires the use of ρ_{ik} units of the auxiliary resource R_i at all times during its execution, where $0 \leqq \rho_{ik} \leqq 1$. At any t, let $r_i(t)$ denote the total amount of the auxiliary resource R_i which is being used at time t. Thus,

$$r_i(t) = \sum_{J_k \in f(t)} \rho_{ik}$$

where f(t) is the set of jobs which are being executed at time t. (The restriction that there are at most n processors can be expressed by requiring $|f(t)| \leqq n$ for all t.)

In this model, the fundamental constraint is simply this:

$$r_i(t) \leqq 1, \quad \text{for all } t \text{ and } i = 1, 2, \ldots , s.$$

In other words, at no time can one use more of any resource than is currently available.

As before, let ω denote the completion time for a set of jobs using an arbitrary priority list and ω_0 denote the optimal completion time for the same set of jobs. Then, we have the following results (see Garey and Graham [5]):

<u>Theorem</u>: For a computing system with n identical processors and one kind of auxiliary resource, we have:

$$\omega/\omega_0 \leqq n.$$

Theorem: For a computing system with $n(\geq 2)$ processors and q auxiliary resources, and for a set of independent jobs, we have:

$$\omega/\omega_0 \leq \min \{ (n + 1)/2, q + 2 - (2q - 1)/n \}$$

In the results presented above, no effort has been spent in obtaining a priority list. A few algorithms, which spend some effort in searching for a priority list, are presented below.

Theorem (Chen & Liu [1]): When Hu's algorithm [7] is applied to schedule a set of jobs with unit execution times on a computing system with n identical processors, then,

$$\omega/\omega_0 \leq 4/3, \quad \text{for } n = 2$$

$$\omega/\omega_0 \leq 2 - 1/(n-1), \quad \text{for } n \geq 3$$

where ω is the completion time for a given set of jobs, when scheduled according to the above algorithm, and ω_0 is the optimal completion time for the same set of jobs.

Theorem (Lam & Sethi [12]): When Coffman and Graham's algorithm [2] is applied to schedule a set of jobs with unit execution times on a computing system with n identical processors, then,

$$\omega/\omega_0 \leq 2 - 2/n.$$

The following two results regarding scheduling a set of independent jobs on a computing system with n identical processors are due to Graham [6].

Theorem: If jobs are scheduled according to a priority list which assigns higher priority to a job with longer execution time, then,

$$\omega/\omega_0 \leq 4/3 - 1/3n.$$

Theorem: If k jobs with longest execution time in the set are scheduled in such a way that the total execution time (for the execution of these k jobs) is minimum, and the other jobs in the set are scheduled according to the rule that whenever a processor is free an arbitrarily chosen job will be executed on that processor, then,

$$\omega/\omega_0 \leq 1 + (1 - 1/n)/(1 + \lfloor k/n \rfloor).$$

One can consider algorithms that produce schedules which are as close to optimal schedules as it is desired at the expense of computation time. An algorithm is said to be an ε-approximation algorithm if for a given ε, the algorithm yields a schedule such that the ration $(\omega - \omega_0)/\omega_0$ is less than ε, where ω is the completion time according to the algorithm and ω_0 is the minimum completion time. Sahni [15] considered the problem of scheduling a set of m independent jobs on a computing system with n identical processors and obtained an ε-approximation algorithm with a complexity $O(m(m^2/\varepsilon)^{n-1})$.

1.4 Scheduling to Meet Deadlines

The question of looking for a schedule to minimize the completion time can be inverted. Suppose there is an unlimited supply of processors. For a fixed deadline, how do we schedule the jobs so that we can complete the given set of jobs within the deadline using a minimal number of processors? Although no results are available for a general set of jobs, there are some interesting results for the case in which the jobs are independent. When the jobs are independent, the problem can be rephrased as a bin-packing problem in which we have an infinite supply of bins of a fixed size and a set of packages to be packed into bins so that the sum of the sizes of the packages in a bin

does not exceed the size of the bin. Unfortunately, this problem also turns out to be NP-hard. Consequently, attention has been directed toward simple algorithms that do not use excessive amounts of resources. Some heuristic algorithms are presented here. Let B_1, B_2, ..., be the bins available. A heuristic algorithm known as the 'next-fit' algorithm is defined as follows: Packages are placed in the bins one by one in some arbitrary order, starting with the bin B_1. If a package does not fit in bin B_i, it is placed in bin B_{i+1}, and no more packages are placed in bin B_i. Another heuristic algorithm, known as the 'first-fit' algorithm is defined as follows: Packages are placed in the bins one by one in some arbitrary order. A package is placed in the bin with the lowest index in which it fits. The first-fit algorithm can be modified as follows: Instead of placing the packages into bins in an arbitrary order, place them in the order of non-increasing package-sizes. This algorithm is known as 'first-fit decreasing' algorithm. Some results due to Johnson et al [8] about these algorithms are given below. Let N_A denote the number of bins needed by algorithm A, and let N_0 be the minimum number of bins needed.

Theorem: For the next-fit (NF) algorithm:

$$\lim_{N_0 \to \infty} N_{NF}/N_0 \leqq 2$$

Theorem: For the first-fit (FF) algorithm:

$$\lim_{N_0 \to \infty} N_{FF}/N_0 \leqq 17/10$$

Theorem: For the first-fit decreasing (FFD) algorithm:

$$\lim_{N_0 \to \infty} N_{FFD}/N_0 \leqq 11/9.$$

CHAPTER 2

SCHEDULING PERIODIC-TIME-CRITICAL JOBS

2.1 Introduction

In the model described in Chapter 1, it is assumed that all jobs are available for execution simultaneously. In some real-time applications, a computer is often required to execute certain jobs in response to some external signals and to guarantee that each such job is completely executed within a specified interval of time following the initiation of the signal that caused the execution of the job. Thus our model can be further generalized by assuming that each job has an <u>initiation</u> or <u>ready time</u>, a time at or after which execution of the job can begin, and a <u>deadline</u>, a time at or prior to which execution of the job must be completed. The scheduling problem in this case is to produce a schedule according to which all jobs can be executed to meet their deadlines. There are many important applications in real-time control or monitoring systems, such as a process control system, in which jobs consist of computations that are executed periodically, with each computation of the job having fixed deadline for completion. The deadline for a given computation can be no later than the time of the request for executing the next computation of the job. Formally, such jobs consist of an infinite sequence of <u>requests</u> with each request having rigid timing bound, i.e., associated with each request there is an initiation time and a deadline for its completion.

Service within this span of time must be guaranteed. An environment, which requires guaranteed service within specified time bounds, is characterized as a "hard" real-time environment. This is quite in contrast to what is called a "soft" real-time environment, where response to a request within a "reasonable" amount of time is considered acceptable. Whereas in a "hard" real-time environment a failure to meet the prescribed deadline results in an irreparable damage, in the "soft" real-time environment, small variations in service are considered tolerable.

A job in a real-time environment which consists of an infinite sequence of requests at a fixed interval of time will be referred to as periodic-time-critical job. A periodic-time-critical job can be specified as a quadruple (C, T, t, t_0), with $C < t \leq T$, where C is the computation time or run-time for each request of the job, t_0 is the time when the first request of the job occurs, and T is the time interval at which successive requests of the job occur. The deadline for the k^{th} request of a job is $t_0 + (k - 1)T + t$. We shall assume that all jobs are independent. As mentioned earlier, the scheduling problem for such a set of jobs is to produce a schedule according to which all requests of all jobs in the set can be executed to meet their respective deadlines. Such a schedule is called a feasible schedule. A set of jobs is said to be schedulable if there exists a feasible schedule for the set of jobs. A set of jobs is said to be schedulable by a scheduling algorithm if the algorithm yields a feasible schedule for the set. Consequently, a scheduling algorithm is said to be optimal, if it

produces a feasible schedule for every schedulable set of jobs. If a scheduling algorithm is not optimal, one would wish to measure the effectiveness of the algorithm in terms of the fraction of schedulable sets of jobs it is capable of feasibly scheduling.

The problem of devising scheduling algorithms for this class of problems has also attracted the attention of many researchers. Scheduling algorithms considered for this problem have been restricted to preemptive priority driven algorithms. This means that whenever there is a request for a job of priority higher than that of the one currently being executed, the running job is immediately interrupted and the newly requested job is started. The interrupted request is resumed later from the point it was interrupted. It is assumed that the time required to switch a processor from one request to another is zero. This assumption is not unrealistic since the switching time is comparatively very small. At any instant a request is said to be _active_ if it has been initiated on or before that time but has not yet been executed.

The specification of preemptive priority-driven scheduling algorithms amounts to the specification of the method of assigning priorities. These algorithms may be classified into two categories according to the method of priority assignment - _static_ or _fixed_ priority algorithms, and _dynamic_ priority algorithms. A _fixed_ priority algorithm is one in which priorities are assigned to jobs and all requests of a job of higher priority have precedence over all requests of a job of lower priority. In a _dynamic_ priority algorithm, priorities are assigned to requests of jobs.

2.2 Previous Work on Time-Critical Jobs

Most of the results in this area are restricted to the case where there is only one processor in the computing system. A dynamic priority algorithm, known as relative urgency algorithm or deadline driven algorithm, assigns priorities to requests according to their deadline, with highest priority being assigned to a request whose deadline is the earliest of all active requests. It has been shown that the relative urgency algorithm is optimal for scheduling a set of jobs on a single processor computing system, in the sense that if there exists a feasible schedule for any given set of jobs then the schedule produced by the relative urgency algorithm is also feasible (see Labetoulle [10]).

Another example of a dynamic priority algorithm is what is called the "e algorithm" (Labetoulle [11]). This algorithm is defined for a multiprocessor computing system as follows.

At any time t, let $C_i(t)$ represent the non-executed portion of an active request of job J_i with deadline d_i. Let $\alpha_i(t) = d_i - C_i(t)$. Clearly, if at any instant t, $\alpha_i(t) > t$, the deadline of job J_i will not be respected.

A switching point is any instant of time where the schedule may be modified.

At a switching point t, a processor is assigned to the active request of J_k, if at time t:

(a) $\alpha_k(t) \leq \alpha_i(t)$ for all i such that job J_i has not been assigned a processor at time t.

and (b) $C_k(t) \leq C_r(t)$ for every r, such that job J_r has an active request, has not been assigned any processor and $\alpha_k(t) = \alpha_r(t)$.

If there are any ties they are broken arbitrarily.

A request which is assigned a processor at time t, continues to be processed till the next switching point, which is the earliest of the following times:

(i) $t + C_k(t)$,

(ii) the time of a new request,

and (iii) $t + \min \{ \alpha_i(t) \mid J_i$ is not being executed on any processor at t $\} - \max \{ \alpha_r(t) \mid J_r$ is being executed at t $\}$.

At any instant, reassignment of processors takes place according to steps (a) and (b).

It has been shown that the "e algorithm" is optimal for a single-processor computing system (see Labetoulle [11]). Nothing has been proved concerning the behavior of "e algorithm" for a multiprocessor computing system.

The above two results regarding the deadline driven algorithm and the "e algorithm" are also applicable to the problem of scheduling a set of independent jobs with individual initiation time and deadline on a computing system with a single processor where preemptions are allowed (Labetoulle [11]).

2.3 Periodic-Time-Critical Jobs

An interesting variation of the above model is obtained

by stipulating that the deadline of each request of a job coincides with the next request of the same job, and by assuming that the first request of all jobs occurs at the same time. A job J in this model can thus be specified by an ordered pair (C, T), where C is the computation time for each request of the job and T is the time interval at which the requests of the job occur. The reciprocal of the request period T will be called the request rate of J. In what follows we shall be concerned with the scheduling of a set of jobs of the type just described. It will further be assumed that all jobs are independent. A set of m jobs will be denoted as $\{J_i \equiv (C_i, T_i)\}_{i=1}^{m}$.

The utilization factor of job $J \equiv (C, T)$ is defined to be the ratio C/T and the utilization factor of a set of jobs is the sum of the utilization factors of all jobs in the set. Notice that the utilization factor of a job is the fraction of the processor time taken up by the job. Clearly, if the utilization factor of a set of jobs is greater than n, it will be impossible to find a schedule that will feasibly schedule the set of jobs on n-processor computing system. Hence a necessary condition for the existence of a feasible schedule for a set of jobs on a computing system with n processors is that the utilization factor of the set of jobs be less than or equal to n. Whether this condition is sufficient or not will depend on the particular scheduling algorithm used.

A set of jobs is said to fully utilize a processor according to a certain scheduling algorithm, if the set of jobs can be feasibly scheduled by that algorithm and increasing the computation time of any one of the jobs in the set would cause the algorithm not to

produce a feasible schedule for the set. For a given algorithm, the minimum achievable processor utilization is the minimum of the utilization factor over all job sets that fully utilize the processor. This means that any set of jobs whose utilization factor is less than or equal to the minimum achievable utilization can always be feasibly scheduled by the corresponding algorithm. (Sets of jobs with larger utilization factor may or may not be feasibly scheduled by the corresponding algorithm.) Thus a possible measure of the "effectiveness" of a scheduling algorithm is the minimum achievable processor utilization of the algorithm. Other things being equal, an algorithm which has higher minimum achievable processor utilization is naturally more "effective" since it enlarges the set of feasibly schedulable job sets.

The problem of scheduling a set of periodic-time-critical jobs on a single-processor computing system was first studied by Liu & Layland [13] and Serlin [16]. Algorithms considered by Liu & Layland fall in the class of premptive priority-driven scheduling algorithms. The deadline driven algorithm, which assigns priorities to the requests according to their deadlines with the highest priority being assigned to the request whose deadline is the earliest, was also considered by Liu & Layland. They obtained the following result:

Theorem: A necessary and sufficient condition that a set of periodic-time-critical jobs be feasibly scheduled on a single processor by the deadline driven scheduling algorithm is that the utilization factor of the set of jobs is less than or equal to 1.

This theorem, together with the fact that for a set of jobs to be feasibly scheduled on a single processor computing system by any

algorithm, its utilization factor should be less than or equal to 1, establishes that the deadline driven scheduling algorithm is optimal, which is in conformity with Labetoulle's result stated earlier.

Another algorithm considered by Liu & Layland was called the rate-monotonic priority assignment algorithm. This algorithm was referred to as the intelligent fixed priority algorithm by Serlin [16]. According to this algorithm, priorities to jobs are assigned in the decreasing order of their request rates. A job with a higher request rate is assigned higher priority over a job with lower request rate. This algorithm is an example of a fixed priority scheduling algorithm. Some results about this algorithm are as follows.

Theorem [13]: Among all fixed priority scheduling algorithms for scheduling a set of periodic-time-critical jobs on a single processor, the rate-monotonic scheduling algorithm is a best one in the sense that if a set of periodic-time-critical jobs can be feasibly scheduled by any fixed assignment of priorities, then this set can also be feasibly scheduled by the rate-monotonic scheduling algorithm.

The following theorem gives a measure of the 'effectiveness' of the rate-monotonic scheduling algorithm.

Theorem A [13]: A set of m periodic-time-critical jobs can be feasibly scheduled on a single processor computing system by the rate-monotonic scheduling algorithm, if the utilization factor of the set is less than or equal to $m(2^{1/m} - 1)$. Also, this bound is tight in the sense that for each m, there exists a set of m jobs with utilization factor $m(2^{1/m} - 1)$ which fully utilizes the processor.

It should be noted that this theorem provides only a sufficient condition for a set of m jobs to be feasibly scheduled by the rate-

scheduling algorithm. Sets of m jobs with utilization factor greater than $m(2^{1/m} - 1)$ may or may not be feasibly scheduled by the rate-monotonic scheduling algorithm. If the request periods $T_1 \leq T_2 \leq \ldots \leq T_m$ of a set of m jobs are such that $T_{i-1} | T_i$ for i = 2, 3, .. , m, then the set of jobs can be feasibly scheduled by the rate-monotonic scheduling algorithm if its utilization factor is less than or equal to 1.

In case m = 2, the result of the last theorem can be strengthened as follows:

Theorem B [16]: Let (C_1, T_1) and (C_2, T_2) with $\lfloor T_2/T_1 \rfloor = r$ be a set of two jobs. If the utilization factor of this set of jobs is less than or equal to $2(\sqrt{r(r-1)} - 1)$, then the set can be feasibly scheduled on a single processor by the rate-monotonic scheduling algorithm.

Liu & Layland also considered an algorithm which is a combination of the rate-monotonic scheduling algorithm and the deadline driven scheduling algorithm. This algorithm is referred to as 'mixed scheduling algorithm'. According to this algorithm, k jobs that have the k shortest request periods, are scheduled according to the rate-monotonic scheduling algorithm, and the remaining jobs are scheduled according to the deadline driven scheduling algorithm when the processor is not occupied by the k jobs that have the k shortest request periods. However, no closed form expression for the minimum achievable processor utilization was found.

As pointed out above, according to the rate-monotonic scheduling algorithm, the minimum achievable processor utilization for a set of m jobs on a single processor is $m(2^{1/m} - 1)$. This result can be refined. The refinement is carried out in two ways - one when the utilization factor of the job with the smallest request

period is fixed, and the other when the utilization factor of m - 1 jobs with the shortest m - 1 request periods is fixed.

Theorem 1: Let $\{J_i \equiv (C_i, T_i)\}_{i=1}^{m}$ be a set of m jobs with $T_1 \leq T_2 \leq \ldots\ldots \leq T_m$. Let $x_1 = C_1/T_1$. If the utilization factor of the set of jobs $J_2, J_3, \ldots\ldots, J_m$ is less than or equal to $(m - 1)\{(2/(1 + x_1))^{1/(m-1)} - 1\}$, then the set can be feasibly scheduled by the rate-monotonic scheduling algorithm.

Proof: It will be sufficient to show that the least upper bound to the utilization factor for the set of m jobs in which the utilization factor of the job with the shortest request period is x_1, is $x_1 + (m - 1)\{(2/(1 + x_1))^{1/(m-1)} - 1\}$.

It was shown in [13] that in determining the least upper bound to the utilization factor according to the rate-monotonic scheduling algorithm, it is sufficient to consider all sets of jobs in which the ratio T_m/T_1 is less than or equal to 2, where T_m is the longest request period and T_1 is the shortest request period.

Let $C_1' = C_1$ and $T_1' = T_1$. Let $C_2', C_3', \ldots\ldots, C_{m-1}'$ be such that $C_1' + C_2' + \ldots\ldots + C_{m-1}' \leq T_1'$. Let $J_i' \equiv (C_i', T_i')$, where $T_{i+1}' = T_i' + C_i'$, for $i = 1, 2, \ldots\ldots, m-2$. Then, if we choose $J_m' \equiv (C_m', T_m')$, where

$$T_m' = T_{m-1}' + C_{m-1}'$$

and

$$C_m' = T_m' - 2(C_1' + C_2' + \ldots\ldots + C_{m-1}')$$

the processor is fully utilized. It was shown in [13] that among all job sets which fully utilize the processor, a job set for which $C_i = T_{i+1} - T_i$, for $i = 1, 2, \ldots\ldots, m-1$, and $C_m = T_m - 2\sum_{i=1}^{m-1} C_i$

has the minimum utilization factor. The utilization factor of the set of jobs $\{J_i' \equiv (C_i', T_i')\}_{i=1}^{m}$ is

$$U = \frac{C_1'}{T_1'} + \frac{C_2'}{T_1' + C_1'} + \ldots\ldots\ldots + \frac{C_{m-1}'}{T_1' + C_1' + \ldots\ldots + C_{m-2}'} + \frac{T_1' - C_1' - \ldots\ldots\ldots C_{m-1}'}{T_1' + C_1' + \ldots\ldots\ldots + C_{m-1}'}$$

Writing x_i for C_i'/T_i', we can write

$$U = x_1 + x_2/(1 + x_1) + \ldots\ldots\ldots + x_{m-1}/(1 + x_1 + \ldots\ldots + x_{m-2})$$
$$+ (1 - x_1 - x_2 \ldots\ldots - x_{m-1})/(1 + x_1 + \ldots\ldots\ldots x_{m-1}) \quad (2.1)$$

This expression for U must be minimized over x_i's, i = 2, 3, ..., m-1, to find the least upper bound to the utilization factor. On setting the first derivatives of U with respect to x_2, x_3, ..., x_{m-1} to zero, we obtain:

$$\partial U/\partial x_2 = 1/(1 + x_1) - x_3/(1 + x_1 + x_2)^2 - \ldots\ldots\ldots -$$
$$x_{m-1}/(1 + x_1 + \ldots\ldots + x_{m-2})^2 - 2/(1 + x_1 + \ldots\ldots x_{m-1})^2 = 0$$

$$\partial U/\partial x_3 = 1/(1 + x_1 + x_2) - x_4/(1 + x_1 + x_2 + x_3)^2 - \ldots\ldots\ldots$$
$$- x_{m-1}/(1 + x_1 + \ldots\ldots + x_{m-2})^2$$
$$- 2/(1 + x_1 + \ldots\ldots + x_{m-1})^2 = 0$$

.
.
.

$$\partial U/\partial x_{m-1} = 1/(1 + x_1 + x_2 + \ldots\ldots + x_{m-2})$$

$$- \; 2/(1 + x_1 + x_2 + \ldots\ldots + x_{m-1})^2 = 0$$

Solving these equations for x_2, x_3, ..., x_{m-1} and substituting these values in (2.1), we obtain

$$U = x_1 + (m - 1)\{(2/(1 + x_1))^{1/(m-1)} - 1\}$$

which proves the assertion.

Note that as m approaches infinity, the utilization factor U approaches $x_1 + \ln(2/(1 + x_1))$.

Also if $x_1 = 0$, the minimum achievable utilization factor for a set of m - 1 jobs is $(m - 1)(2^{1/(m-1)} - 1)$, which is in conformity with the result of Theorem A.

Figure 2.1 shows how the minimum achievable utilization factor of a set of jobs varies as a function of the utilization factor of the job with the shortest request period.

Suppose we are given a set of m - 1 jobs which can be feasibly scheduled according to the rate-monotonic scheduling algorithm, but do not fully utilize the processor. It will be useful at times to know whether a job can be added to this set so that the new set can also be feasibly scheduled. The result of Theorem A can be applied to determine the least upper bound to the utilization factor that the new job can have. However, if the request period of the new job is longer than the request period of any other job in the set, then the following theorem gives a stronger result.

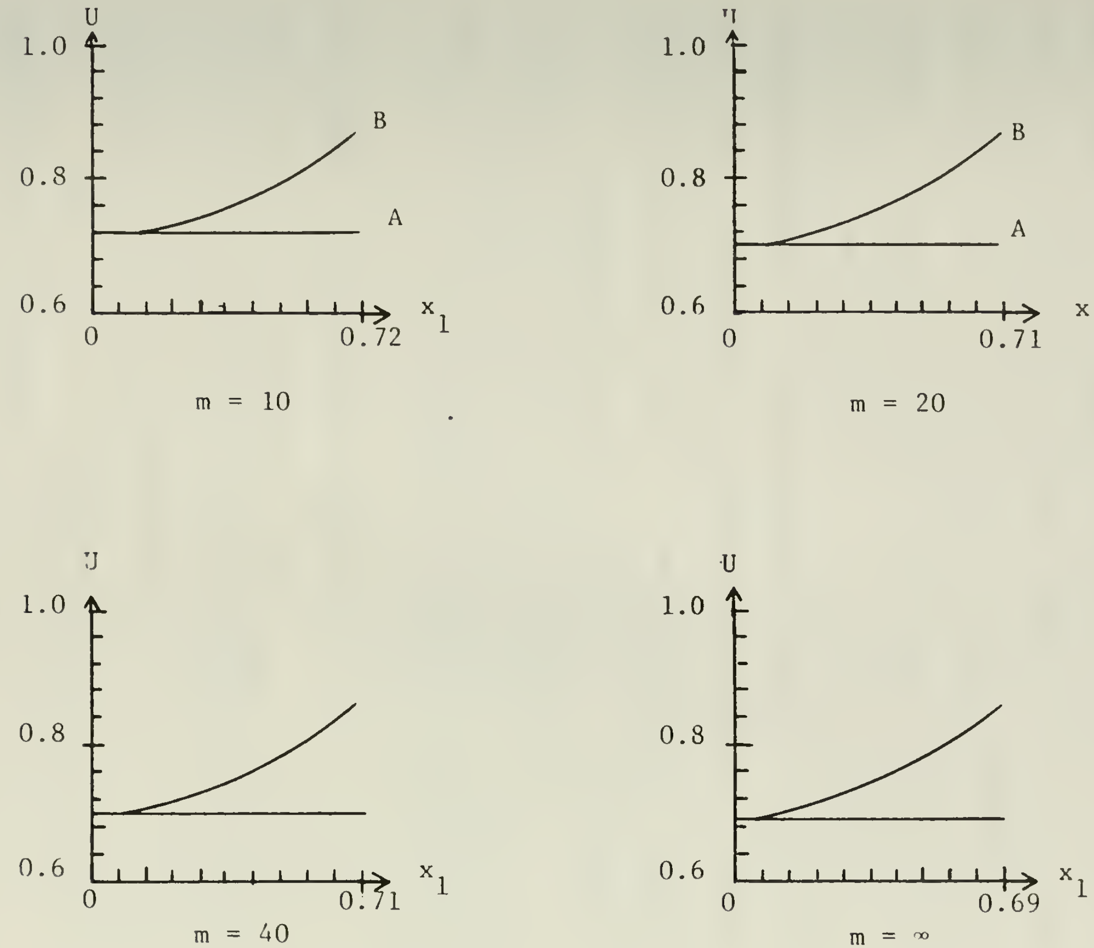

Figure 2.1 Curves A and B give the minimum utilization factor for a set of m jobs according to Theorem A and Theorem 1, respectively.

Theorem 2: Let $\{J_i \equiv (C_i, T_i)\}_{i=1}^{m}$ be a set of m jobs with $T_1 \leqq T_2 \leqq \ldots\ldots \leqq T_m$. Let the utilization factor of the set of jobs $J_1, J_2, \ldots\ldots, J_{m-1}$ be $u = \sum_{i=1}^{m-1}(C_i/T_i) \leqq (m-1)(2^{1/(m-1)} - 1)$.

If C_m/T_m is less than or equal to $2\{1 + u/(m-1)\}^{-(m-1)} - 1$, then the given set can be feasibly scheduled by the rate-monotonic scheduling algorithm.

Proof: It is sufficient to determine the least upper bound to the utilization factor of a set of m jobs, in terms of u, the utilization factor of the m - 1 jobs with the m -1 shortest request periods.

From the given set of m - 1 jobs $\{J_i \equiv (C_i, T_i)\}_{i=1}^{m-1}$, we construct another set of m - 1 jobs $\{J_i' \equiv (C_i', T_i')\}_{i=1}^{m-1}$ as follows:

Let

$$C_1' = C_1$$

$$T_1' = T_1$$

$$C_2' = (T_1' + C_1')C_2/T_2$$

$$T_2' = T_1' + C_1'$$

$$\cdot$$

$$\cdot$$

$$C_{m-1}' = (T_{m-2}' + C_{m-2}')C_{m-1}/T_{m-1}$$

$$T_{m-1}' = T_{m-2}' + C_{m-2}'$$

The condition $u \leqq (m-1)(2^{1/(m-1)} - 1)$ guarantees that

$C_1' + C_2' + \ldots\ldots + C_{m-1}' \leqq T_1'$.

Let $T_m' = T_{m-1}' + C_{m-1}'$ and $C_m' = T_1' - \sum_{i=1}^{m-1} C_i'$. Then

the set of jobs $\{J_i' \equiv (C_i', T_i')\}_{i=1}^{m}$ fully utilizes the processor.

Also, since the utilization factor of $J_i' = C_i/T_i$ = utilization factor of J_i, for $i = 1, 2, \ldots\ldots, m-1$, this set of jobs is a member of the class of sets for which the utilization factor of the m-1 jobs with m-1 shortest request periods is u. Further, among all sets of m jobs having C_1', T_1', T_2', ..., T_m' as in this case, the set of jobs J_1', J_2', ... , J_m' has the minimum utilization factor. The utilization factor for this set of jobs is:

$$U = C_1'/T_1' + C_2'/(T_1' + C_1') + \ldots\ldots +$$

$$(T_1' - C_1' - \ldots - C_{m-1}')/(T_1' + C_1' \ldots + C_{m-1}').$$

In order to find the minimum utilization factor over all possible sets of values C_1', T_1', T_2', ..., T_m', we have to minimize the above expression for U.

Let $x_i = C_i'/T_1'$, for i - 1, 2, ..., m-1.

Then,

$$U = x_1 + x_2/(1 + x_1) + \ldots\ldots\ldots + x_{m-1}/(1 + x_1 + \ldots\ldots\ldots + x_{m-2}) + (1-x_1 - \ldots - x_{m-1})/(1 + x_1 + \ldots + x_{m-1})$$

or,

$$U = u + (1 - x_1 - \ldots - x_{m-1})/(1 + x_1 + \ldots + x_{m-1}) \qquad (2.2)$$

where,

$$u = x_1 + x_2/(1 + x_1) + \ldots + x_{m-1}/(1 + x_1 + \ldots + x_{m-2}) \qquad (2.3)$$

Thus we have to minimize the expression for U as given in (2.2) subject to the condition (2.3). This can be done by forming the Lagrangian

$$L = U + \lambda H,$$

where $H = x_1 + x_2/(1 + x_1) + .. + x_{m-1}/(1 + x_1 + .. + x_{m-2}) - u = 0$

and then minimizing the function L over x_i's. The minimum of this function is obtained by solving the following set of equations:

$$\partial L/\partial x_1 = -2/(1 + x_1 + \ldots + x_{m-1})^2 + \lambda\{1 - x_2/(1 + x_1)^2 - \ldots$$

$$\ldots. - x_{m-1}/(1 + x_1 + \ldots + x_{m-2})^2\} = 0$$

$$\partial L/\partial x_2 = -2/(1 + x_1 + \ldots + x_{m-1})^2 + \lambda\{1/(1 + x_1)$$

$$- x_3/(1 + x_1 + x_2)^2 - \ldots\ldots.$$

$$- x_{m-1}/(1 + x_1 + \ldots.. + x_{m-2})^2\} = 0$$

..

..

$$\partial L/\partial x_{m-2} = -2/(1 + x_1 + \ldots.. + x_{m-1})^2 + \lambda\{1/(1 + x_1 + \ldots + x_{m-3})$$

$$- x_{m-1}/(1 + x_1 + \ldots + x_{m-2})^2\} = 0$$

$$\partial L/\partial x_{m-1} = -2/(1 + x_1 + \ldots + x_{m-1})^2 - \lambda/(1 + x_1 + .. + x_{m-2}) = 0$$

$$\partial L/\partial \lambda = x_1 + x_2/(1 + x_1) + \ldots + x_{m-1}/(1 + x_1 + .. + x_{m-2}) - u = 0$$

Solving these equations for x_i's and λ, we obtain

$$x_i = \left(\frac{2}{\lambda}\right)^{\frac{i-1}{m}} \left[\frac{2}{\lambda}^{\frac{1}{m}} - 1 \right]$$

for i = 1, 2, ... , m-1.

$$\lambda = \frac{2}{\left[\frac{u}{m-1} + 1\right]^m}$$

Substituting the values of x_i's in (2.2), we obtain

$$U = u + 2\left(1 + \frac{u}{m-1}\right)^{-(m-1)} - 1,$$

which means that so long as the utilization factor of the m^{th} job with request period $T_m \geq T_{m-1}$ is less than or equal to $2\{1 + u/(m-1)\}^{-(m-1)} - 1$, it is possible to schedule all jobs on one processor according to the rate-monotonic scheduling algorithm.

Note that as m approaches infinity, the quantity $2\{1 + u/(m-1)\}^{-(m-1)} - 1$ approaches 2 exp(-u) - 1 in the limit.

Figure 2.2 compares the minimum utilization factor of the m^{th} job that can be obtained by applying the result of Theorem A and that of the above theorem.

As was pointed out earlier, it has been shown that amongst all fixed priority scheduling algorithms, the rate-monotonic scheduling algorithm is a best one in the sense that if a given set of jobs can be feasibly scheduled according to any fixed priority scheduling algorithm, that set can also be feasibly scheduled by the rate-monotonic scheduling algorithm. It would, however, be interesting to investigate how some other fixed priority scheduling

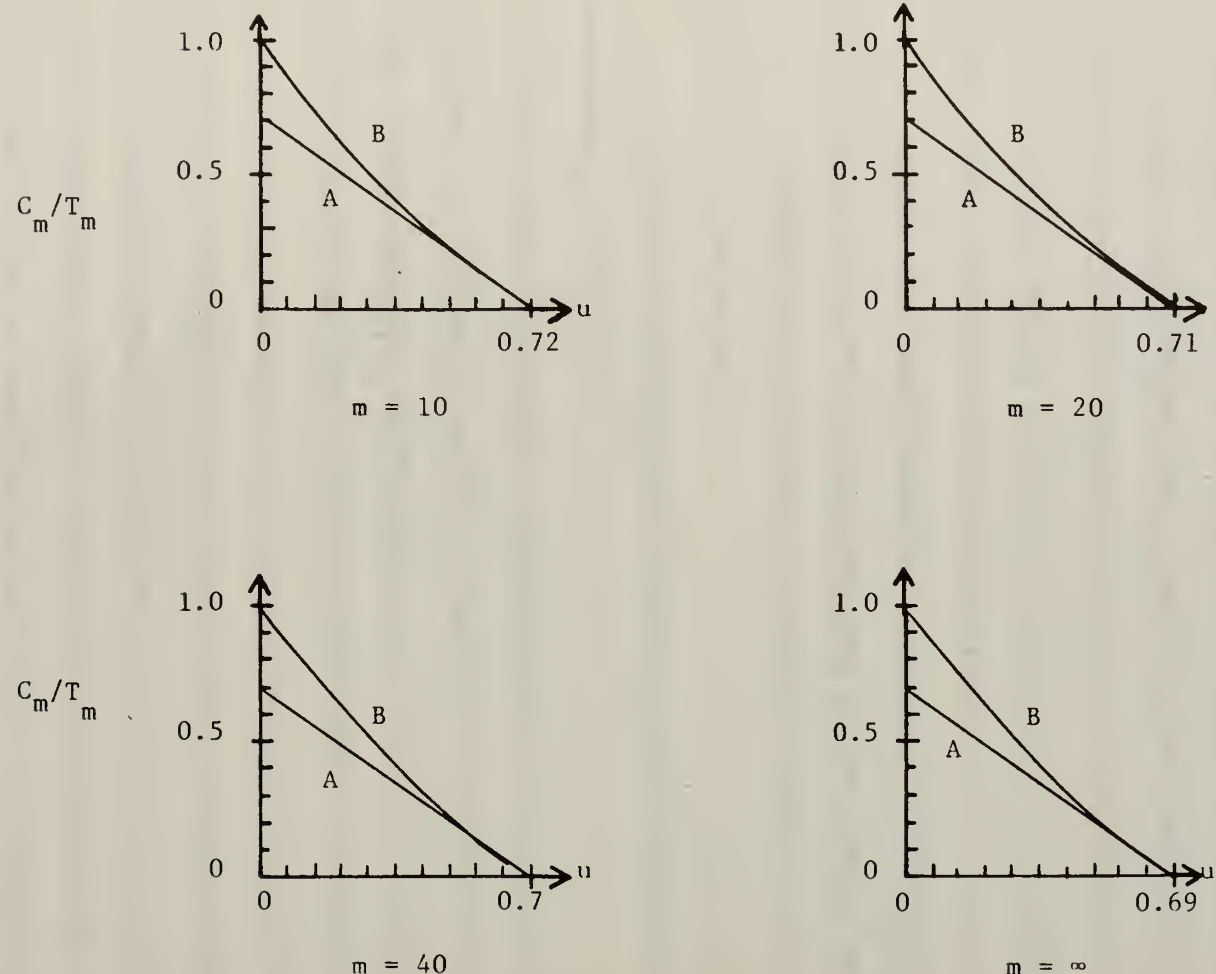

Figure 2.2 Curves A and B give the minimum utilization factor for the m^{th} job according to Theorem A and Theorem 2, respectively.

algorithms perform. We have investigated the performance of another heuristic fixed priority algorithm in the next section.

2.4 Reverse-Rate-Monotonic Priority Assignment Algorithm

In the reverse-rate-monotonic priority assignment algorithm, priorities to jobs are assigned on the basis of increasing order of their request rates. That is, a job with a smaller request rate is assigned higher priority over a job with larger request rate. We note that the assignment of priorities in this case is exactly in the reverse order of those in the case of rate-monotonic scheduling algorithm (hence the name), and, not unexpectedly, turns out to be a worst possible priority assignment, that is, if a set of jobs can be feasibly scheduled according to reverse-rate monotonic priority assignment, then it can also be feasibly scheduled by any fixed assignment of priorities.

Before we take up the analysis of this algorithm, we prove the following result.

Theorem 3: If the sum of computation times of a set of jobs is less than or equal to the shortest request period amongst all jobs in the set, then the set of jobs can be feasibly scheduled by any 'fixed' priority scheduling algorithm.

Proof: Let $\{J_i \equiv (C_i, T_i)\}_{i=1}^{m}$ be a set of m jobs, with $T_1 \leq T_2 \leq \ldots\ldots \leq T_m$ and let $C_1 + C_2 + \ldots\ldots + C_m \leq T_1$.

Let J_i be the job that is assigned the lowest priority. Since the execution of the request of a higher priority job will not be interrupted by a request of lower priority job, it will be sufficient to show that each request of the lowest priority job, J_i,

is completed before the corresponding deadline.

Suppose the first k requests of job J_i are all completed before the respective deadlines, and that the $(k + 1)^{th}$ request at time kT_i is not completed before its deadline. We will show that this is not possible.

If the request of job J_i is not completed before time $(k+1)T_i$, then the processor is busy throughout the time interval $[kT_i, (k+1)T_i]$. Suppose the J_i request at time $(k-1)T_i$ is completed at time $t_o \varepsilon ((k-1)T_i, kT_i]$. Since J_i has the lowest priority, no other job can have an active request at time t_0 (unless, of course, it is initiated at time t_0). There must be some job which has at least two of its requests initiated in the time interval $[t_0, (k+1)T_i]$, because otherwise, since the sum of computation times of all jobs is less than or equal to $T_1 \leq (k+1)T_i - t_0$, the request of job J_i at time kT_i must have been completed on or before the time $(k+1)T_i$. We consider two cases:

(i) All requests initiated during the time interval $[t_0, kT_i]$ are completed on or before time kT_i.

(ii) Some of the requests initiated in the time interval $[t_0, kT_i]$ are still active at time kT_i.

In case (i), if there is no job which has more than one request initiated in the interval $[kT_i, (k+1)T_i]$, since the sum of computation times of all jobs is less than or equal to the shortest request period, which in turn is less than or euqal to T_i, the J_i request initiated at time kT_i must be completed on or before time $(k+1)T_i$. So, suppose there are jobs which have two or more requests

initiated in the time interval $[kT_i, (k+1)T_i]$. Let J_r be the job whose second request is initiated earlier than the second request of any other job. In the time interval $[pT_r, (p+1)T_r]$, where pT_r is the time when the first of requests of job J_r is initiated in the time interval $[kT_i, (k+1)T_i]$, no job has more than one request initiated. Moreover, the processor is busy throughout the time interval $[kT_i, (p+1)T_r]$. The length of this time interval is greater than or equal to T_1, which is greater than or equal to the sum of the computation times of all jobs. Hence the request of job J_i at time kT_i must be completed on or before time $(p+1)T_r \leq (k+1)T_i$.

In case (ii), since there are one ore more requests active at time kT_i, the processor must be busy at least from the time the first of these active requests was initiated in the time interval $[t_o, kT_i)$ upto the time kT_i. Let $t_1 \varepsilon [t_0, kT_i]$ be the earliest time, such that the processor is busy throughout the time interval $[t_1, kT_i]$. Amongst all jobs that have two or more requests initiated in the time interval $[t_1, (k+1)T_i]$, let J_r be the one whose second request is initiated at the earliest. Suppose this second request is initiated at time $p'T_r > kT_i$. Then, since each job has at most one request in the time interval $[t_1, p'T_r]$, of length greater than or equal to $T_r \geq T_1$, all requests initiated in this time interval, including that of J_i at kT_i, must be executed on or before $p'T_r \leq (k+1)T_i$. On the other hand, if $p'T_r \leq kT_i$, since there is at most one request initiated for each job in the time interval $[t_1, p'T_r]$, and since the length of this time interval is greater than or equal to T_1, all requests initiated in this interval must be

executed on or before $p'T_r$. We shift our t_1 to $p'T_r$ and repeat the above arguments for the new interval $[t_1, (k+1)T_i]$. By repititive application of the above arguments, if necessary, we see that the request of job J_i at time kT_i must be completed on or before time $(k+1)T_i$.

For the reverse-rate-monotonic priority assignment algorithm, we have the following result:

Corollary 1: A necessary and sufficient condition for a set of m jobs to be feasibly scheduled according to the reverse-rate-monotonic priority assignment algorithm is that the sum of computation times of all the jobs in the set be less than or equal to the shortest request period in the set.

Proof: Let $\{J_i \equiv (C_i, T_i)\}_{i=1}^{m}$ be a set of m jobs with $T_1 \leqq T_2 \leqq \ldots \leqq T_m$.

We show first the necessity. According to the reverse-rate-monotonic priority assignment algorithm, the job with the shortest request period will have the lowest priority. Suppose the first request of all jobs are initiated simultaneously at time t_0. The first request of job J_1 will, therefore, be executed after the first request of all the jobs have been executed. If the first request of job J_1 is to be executed before its deadline, it is necessary that $C_1 + C_2 + \ldots.. + C_m \leqq T_1$.

We now show sufficiency of the condition. Since the sum of the computation times of all the jobs in the set is less than or equal to the shortest request period, according to Theorem 3, any fixed assignment of priorities, and in particular the reverse-

rate-monotonic priority assignment, will produce a feasible schedule for the given set of jobs.

Corollary 2: Amongst all fixed priority assignment algorithms, the reverse-rate-monotonic priority assignment is a worst one, in the sense that if a set of jobs can be feasibly scheduled according to the reverse-rate-monotonic priority assignment it can also be feasibly scheduled by any fixed priority scheduling algorithm.

Proof: If a given set of jobs can be feasibly scheduled by the reverse-rate-monotonic priority assignment, then the sum of the computation times of all jobs in the set must be less than or equal to the shortest request period in the set. Therefore, any fixed assignment of priorities will produce a feasible schedule for the set.

CHAPTER 3

SCHEDULING PERIODIC-TIME-CRITICAL JOBS ON A MULTIPROCESSOR COMPUTING SYSTEM

3.1 Introduction

Recent progress in hardware technology and computer architecture has led to the design and construction of computer systems that contain a large number of processors. One of the most significant advantages attributed to multiprocessor computing systems is the potential decrease in computation time for a large class of problems achievable by parallel programming, that is, the concurrent execution of independent portions of a computational job. It is, therefore, both of practical and theoretical importance to investigate how to make efficient use of multiprocessor systems for the periodic-time-critical jobs described earlier. In this chapter, the problem of scheduling periodic-time-critical jobs on a multiprocessor computing system is considered.

3.2 Rate-Monotonic and Deadline Driven Scheduling Algorithms for Multiprocessor Computing Systems

One would naturally hope that a simple extension of the algorithms developed for single-processor computing systems would give satisfactory results when applied to scheduling jobs on multiprocessor computing systems. Unfortunately, this does not turn out to be the case, as the following example shows.

A priority-driven preemptive scheduling algorithm for a multiprocessor computing system works as follows. Priorities are assigned to the requests of jobs in the set. At any instant when a processor

is free, it is assigned to an active request of highest priority. Also if a request occurs at an instant when all the processors are assigned, and if this request has priority higher than the priorities of the requests which are being executed at that instant, then this request preempts the request with the lowest priority. Ties are broken arbitrarily. As in the case of a single processor computing system, the rate-monotonic scheduling algorithm assigns priorities to jobs on the basis of their request rates, with higher priority being assigned to a job with higher request rate over a job with lower request rate. The deadline driven scheduling algorithm assigns priorities to requests on the basis of their deadlines with highest priority being assigned to the request whose current deadline is the earliest. Here again ties are broken arbitrarily.

Consider a set of m jobs $\{J_i \equiv (C_i, T_i)\}_{i=1}^{m}$ to be scheduled on n-processor computing system (m > n), where

$$C_i = 2\varepsilon \qquad i = 1, 2, \ldots\ldots, m-1$$

$$T_i = 1, \qquad i = 1, 2, \ldots\ldots, m-1$$

and

$$C_m = 1$$

$$T_m = 1 + \varepsilon.$$

Figure 3.1 is a schedule of this set of jobs on n-processor computing system, according to the rate-monotonic scheduling algorithm. Since jobs $J_1, J_2, \ldots, J_{m-1}$, each with request period 1, have higher priority than that of job J_m with request period $1 + \varepsilon$, processors P_1 through P_n are assigned to the jobs $J_1, J_2, \ldots, J_n$ ($n \leq m - 1$), during the time intervals $[0, 2\varepsilon]$ and $[1, 1 + 2\varepsilon]$. Thus, the maximum

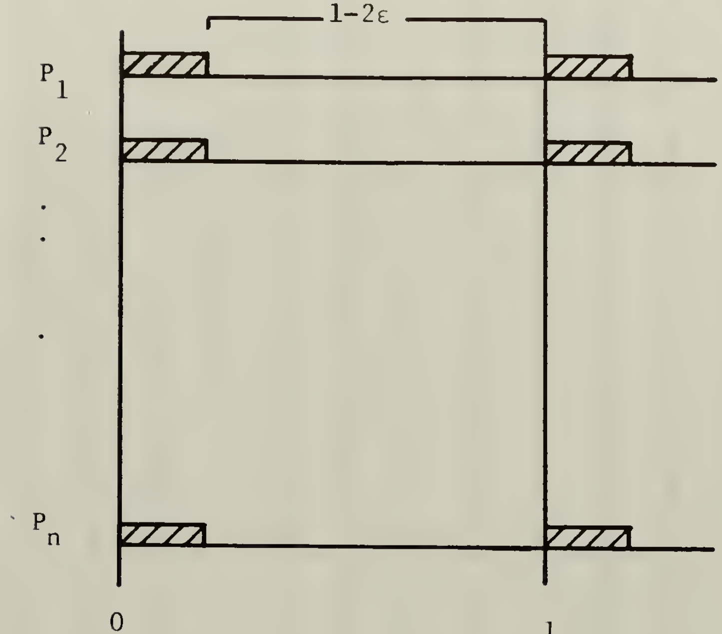

Figure 3.1 A schedule of m jobs on n processors according to the rate-monotonic scheduling algorithm

time available for the first request of job J_m which occurred at time 0, with deadline $1 + \varepsilon$, on any one of the processors is less than or equal to $1 - 2\varepsilon$. Hence the first request of job J_m cannot be completely executed before its deadline, and, therefore, this set of jobs cannot be feasibly scheduled on n-processor computing system according to the rate-monotonic scheduling algorithm. The utilization factor of this set of jobs is $U = 2(m-1)\varepsilon + 1/(1 + \varepsilon)$. As $\varepsilon \to 0$, $U \to 1$. Thus the minimum achievable utilization factor for a set of m jobs on n-processor computing system ($n < m$) according to the rate-monotonic scheduling algorithm is less than or equal to 1.

Similarly, when the above set of jobs is scheduled according to the deadline driven scheduling algorithm, the first requests of jobs $J_1, J_2, \ldots\ldots, J_{m-1}$, with deadline 1 will have higher priority over the first request of job J_m, whose deadline is $1 + \varepsilon$. Consequently, the n processors will be occupied by jobs $J_1, J_2, \ldots\ldots, J_n$, ($n \leqq m-1$) during the time interval $[0, 2\varepsilon]$, leaving a maximum of $1 - \varepsilon$ units of time for the first request of job J_m before its deadline at $1 + \varepsilon$ (figure 3.2). Hence this set of jobs cannot be feasibly scheduled by the deadline driven scheduling algorithm either.

However, if job J_m is assigned the highest priority, then this set of jobs can be feasibly scheduled. This example shows that the rate-monotonic scheduling algorithm when applied to scheduling a set of jobs on a multiprocessor computing system is not optimal amongst all fixed priority scheduling algorithms. Neither is the deadline driven scheduling algorithm optimal amongst all scheduling algorithms, as was the case on a single processor computing system. It is, therefore, desirable to look for better scheduling strategies that will lead to

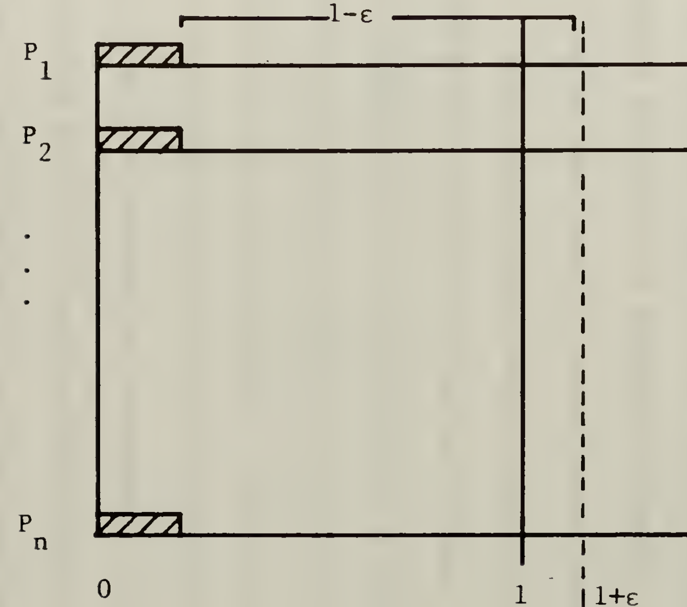

Figure 3.2 A schedule of m jobs on n processors according to the deadline driven scheduling algorithm.

more efficient use of multiprocessor computing systems.

The problem of devising optimal algorithms to schedule a set of periodic time-critical jobs on a fixed number of processors turns out to be a difficult one. An alternative approach is to partition the jobs into groups so that each group of jobs can be feasibly scheduled on a single-processor according to some scheduling algorithm. The scheduling algorithm to be applied to the individual groups in the partition will influence the partitioning process, since each individual group of jobs must be feasibly schedulable on a single processor according to the designated algorithm. The problem is then reduced to determining an optimal partitioning of a given set of jobs with respect to the designated algorithm to be used for the groups in the partition. An _optimal partitioning_ of a set of jobs _with respect to a scheduling algorithm for single processor_ is one that uses a minimum number of processors. Since for a finite set of jobs, there are only a finite number of possible partitions, the problem of determining an optimal partition can be solved by an exhaustive search. However, the number of possible partitions increases exponentially with an increase in the number of jobs. Such an exhaustive search will, therefore, require considerable computation time and will offset the advantage gained by using an optimal partition. Hence, it is interesting to consider some "good" partitioning scheme. By a good partitioning scheme, we mean a scheme under which the number of processors required is reasonably close to the number required by an optimal partitioning.

Recall that if the utilization factor of a set of jobs is less than or equal to 1, then the set can be feasibly scheduled on a single processor computing system according to the deadline driven

scheduling algorithm. The problem of partitioning a set of jobs with respect to the deadline driven scheduling algorithm is then the same as the bin-packing problem where it is desired to pack a set of packages into bins of fixed size so that the sum of the sizes of the packages in a bin does not exceed the size of the bin. This can be seen by imagining a job as a package of size equal to its utilization factor and a processor as a bin of size 1. Thus all results about the bin-packing problem will also be applicable in this case. However, partitioning a given set of jobs with respect to the rate-monotonic scheduling algorithm is no longer the same as the bin-packing problem. An obvious reason is that a set of jobs may not be feasibly schedulable on a single processor according to the rate-monotonic scheduling algorithm, even if the utilization factor of this set of jobs is less than or equal to 1. The total utilization factor of a set of jobs that can be 'accommodated' on a single processor (i.e. can be feasibly scheduled on a single processor) depends not only on the number of jobs in the set, but also on their computation times and request periods. There is, however, a more subtle point that should be noted. A seemingly possible approach, using the result in Theorem A of chapter 2, would be to consider the size of the bins to be ln 2, and apply the known bounds obtained for the bin-packing problem. This approach is, however, false since the utilization factor of a set of jobs that can be feasibly scheduled on a single processor may turn out to be larger than ln 2.

With this background in mind, the problem of finding an algorithm which produces an optimal partitioning for a set of jobs with respect to the rate-monotonic scheduling algorithm, appears to be a rather difficult one. In view of this, it is interesting to investigate into the performance of some 'heuristic' algorithms. In what follows, two such 'heuristic' algorithms have been considered and in both cases, bounds on their worst-case performance relative to the optimal partitioning have been determined.

3.3 The Rate-Monotonic-Next-Fit Scheduling Algorithm

The first algorithm considered is named the rate-monotonic next-fit scheduling algorithm. According to this algorithm, jobs are first arranged in descending order of their request rates. (For convenience, they are renumbered as J_1, J_2, ..., J_m.) The assignment scheme is as follows:

Step 1: Set $i = j = 1$.

Step 2: Assign job J_i to processor P_j, if job J_i together with the jobs that have been assigned to P_j can be feasibly scheduled on P_j according to the rate-monotonic scheduling algorithm. Else, assign J_i to P_{j+1} and set j to $j + 1$. Go to step 3.

Step 3: Set $i = i + 1$, and repeat step 2 until all jobs have been assigned.

The number j so obtained is the number of processors that will be required under this partitioning algorithm.

As an example of the application of the rate-monotonic-next-fit scheduling algorithm, consider the following set of jobs:

(1, 2); (.1, 2.5); (1, 3); (.1, 4.5); (1, 5);

(1, 6); (1, 7); (1, 8); (.1, 8.5); (1, 9).

The application of the above algorithm will produce the following assignment, requiring 4 processors.

Jobs (1, 2), (.1, 2.5) will be assigned to processor P_1;

Jobs (1, 3), (1, 4), and (.1, 4.5) will be assigned to processor P_2;

Jobs (1, 5), (1, 6), (1, 7), (1, 8), (.1, 8.5) will be assigned to processor P_3; and

Job (1, 9) will be assigned to processor P_4.

However, the following partitioning will require only 3 processors, which is the minimum for this set of jobs if the rate-monotonic algorithm is to be used for each group in the partition:

Group 1: (1, 2), (1, 3) and (1, 6).

Group 2: (1, 4), (1, 5), (1, 7), (1, 8) and (1, 9).

Group 3: (.1, 2.5), (.1, 4.5) and (.1, 8.5).

It is interesting to note that since the utilization factor of all the jobs in group 2 and group 3 above is less than 1, all these jobs can be feasibly scheduled on a single processor using the deadline driven scheduling algorithm. Thus partitioning this set of jobs with respect to deadline driven scheduling algorithm requires only 2 processors.

We have the following theorem concerning this algorithm.

Theorem 4: Let N be the number of processors required to feasibly schedule a set of jobs by the rate-monotonic-next-fit

scheduling algorithm, and N_0 the minimum number of processors required to feasibly schedule the same set of jobs, then as N_0 approaches infinity:

$$2.4 \leq \lim_{N_0 \to \infty} N/N_0 \leq 2.67$$

Proof: In order to establish the lower bound, we show that given $\varepsilon > 0$, there exists a set of jobs for which N/N_0 is greater than $2.4 - \varepsilon$. Let N be chosen to be 12k. Consider the set of jobs:

$$(1, 2); (\delta, 2); (1, 3); (\delta, 3); (2, 4); (\delta, 4);$$

$$(2, 6); (\delta, 6); \ldots\ldots\ldots\ldots\ldots;$$

$$(2^{\frac{N}{2}-1}, 2^{\frac{N}{2}}); (\delta, 2^{\frac{N}{2}}); (2^{\frac{N}{2}-1}, 3\times 2^{\frac{N}{2}-1}); (\delta, 3\times 2^{\frac{N}{2}-1}).$$

When we apply the rate-monotonic-next-fit scheduling algorithm to this set of jobs, jobs (1, 2) and $(\delta, 2)$ are assigned to the first processor, jobs (1, 3) and $(\delta, 3)$ are assigned to the second processor, jobs (2, 4) and $(\delta, 4)$ are assigned to the third processor, and so on. Jobs $(2^{N/2-1}, 3\times 2^{N/2-1})$ and $(\delta, 3\times 2^{N/2-1})$ are assigned to the N^{th} processor. Thus total number of processors required for this set of jobs when the rate-monotonic-next-fit scheduling algorithm is applied, is N.

However, this set of jobs can be feasibly scheduled on $5N/12 + 1$ processors. This can be seen as follows. Let us divide the jobs into k + 1 subsets as follows. For i = 1, 2, ..., k, let the i^{th} subset consist of the following jobs:

$$(2^{6(i-1)}, 2\times 2^{6(i-1)}), (2^{6(i-1)}, 3\times 2^{6(i-1)})$$

$$(2^{6(i-1)+1}, 2\times 2^{6(i-1)+1}), (2^{6(i-1)+1}, 3\times 2^{6(i-1)+1}),$$

$$\ldots\ldots \qquad\qquad \ldots\ldots$$

$(2^{6(i-1)+5}, 2\times 2^{6(i-1)+5})$ and $(2^{6(i-1)+5}, 3\times 2^{6(i-1)+5})$.

All jobs with computation time δ form the $(k+1)^{th}$ subset.

Each of the first k of these subsets can be partitioned into five groups as follows:

Group 1: $(2^{6(i-1)}, 2\times 2^{6(i-1)})$ and $(2^{6(i-1)+1}, 2\times 2^{6(i-1)+1})$.

Group 2: $(2^{6(i-1)+2}, 2\times 2^{6(i-1)+2})$ and $(2^{6(i-1)+3}, 2\times 2^{6(i-1)+3})$.

Group 3: $(2^{6(i-1)+4}, 2\times 2^{6(i-1)+4})$ and $(2^{6(i-1)+5}, 2\times 2^{6(i-1)+5})$.

Group 4: $(2^{6(i-1)}, 3\times 2^{6(i-1)})$, $(2^{6(i-1)+1}, 3\times 2^{6(i-1)+1})$ and $(2^{6(i-1)+2}, 3\times 2^{6(i-1)+2})$.

Group 5: $(2^{6(i-1)+3}, 3\times 2^{6(i-1)+3})$, $(2^{6(i-1)+4}, 3\times 2^{6(i-1)+4})$ and $(2^{6(i-1)+5}, 3\times 2^{6(i-1)+5})$.

Each of these groups can be feasibly scheduled on a single processor according to the rate-monotonic scheduling algorithm. Further, if δ is chosen so small such that $N\delta \leq 2$, then all jobs with computation time δ can be scheduled on one processor. This arrangement will thus take only 5k + 1 processors. Thus $N_0 \leq 5k + 1 = 5N/12 + 1$, and so $N/N_0 \geq 12N/(5N + 12)$. By selecting N to be sufficiently large, we can make this ratio approach the limit 2.4, or, in other words, for a given $\varepsilon > 0$, there is an integer N, such that $N/N_0 > 2.4 - \varepsilon$, thus establishing the lower bound.

We now show that the ratio N/N_0 is upper bounded by 2.67. To this end, we define a function f mapping the utilization factors

of jobs into the reals. Let u be the utilization factor of a job. We define

$$f(u) = \begin{cases} 2.67u & 0 \leq u < 0.75 \\ 2 & u \geq 0.75 \end{cases}$$

Let $J_1, J_2, \ldots\ldots, J_m$ be m jobs with utilization factor $u_1, u_2, \ldots\ldots, u_m$ respectively. Let $J_{p1}, J_{p2}, \ldots\ldots, J_{pk_p}$ be the k_p jobs assigned to the p^{th} processor according to an optimal assignment. Then

$$\sum_{r=1}^{k_p} f(u_{pr}) \leq 2.67 \sum_{r=1}^{k_p} u_{pr} \leq 2.67$$

$$\text{for } p = 1, 2, \ldots\ldots, N_0$$

Summing up over all processors, we have

$$\sum_{i=1}^{m} f(u_i) \leq 2.67N_0 \qquad (3.1)$$

In the case of assignment of processors according to the rate-monotonic-next-fit scheduling algorithm, we will show that if $u_{p1}, u_{p2}, \ldots\ldots, u_{pk_p}$ are the utilization factors of the k_p jobs assigned to the p^{th} processor, either

$$\sum_{r=1}^{k_p} f(u_{pr}) \geq 1$$

or

$$\sum_{r=1}^{k_p} f(u_{pr}) + f(u_{(p+1)1}) \geq 2,$$

where $u_{(p+1)1}$ is the utilization factor of the job which could not be assigned to the p^{th} processor.

Suppose $\sum_{r=1}^{k_p} f(u_{pr}) < 1$ for some p. Since this means that $u_{pr} < 0.75$ for $r = 1, 2, \ldots, k_p$, we have $\sum_{r=1}^{k_p} f(u_{pr}) = 2.67 \sum_{r=1}^{k_p} u_{pr}$. Thus, $\sum_{r=1}^{k_p} u_{pr} < 1/2.67$. Therefore, according to Theorem 2, $u_{(p+1)1} > 2\exp[-\sum_{r=1}^{k_p} u_{pr}] - 1$. If $u_{(p+1)1} \geqq 0.75$, then $f(u_{(p+1)1}) = 2$, and we have $\sum_{r=1}^{k_p} f(u_{pr}) + f(u_{(p+1)1}) > 2$.

Otherwise, we have

$$\sum_{r=1}^{k_p} f(u_{pr}) + f(u_{(p+1)1}) \geqq 2.67[\sum_{r=1}^{k_p} u_{pr} + 2\exp(-\sum_{r=1}^{k_p} u_{pr}) - 1] > 2,$$

since for $x < 1/2.67$, the expression $x + 2\exp(-x) - 1 \geqq 2/2.67$.

Let N be the total number of processors used according to the rate-monotonic-next-fit scheduling algorithm. Let processor p, $1 \leq p < N$, be the first processor for which

$$\sum_{r=1}^{k_p} f(u_{pr}) < 1.$$

Then

$$\sum_{r=1}^{k_p} f(u_{pr}) + \sum_{r=1}^{k_{p+1}} f(u_{(p+1)r}) \geqq 2.$$

We then repeat the above procedure on processors p+2 to N. All this implies that

$$\sum_{p=1}^{N} [\sum_{r=1}^{k_p} f(u_{pr})] \geqq N - 1.$$

Thus

$$\sum_{i=1}^{m} f(u_i) = \sum_{p=1}^{N} \left[\sum_{r=1}^{k_p} f(u_{pr}) \right] \geqq N - 1 \qquad (3.2)$$

Combining the result of inequalities (3.1) and (3.2), we have

$$N - 1 \leqq \sum_{i=1}^{m} f(u_i) \leqq 2.67N_0$$

or

$$\lim_{N_0 \to \infty} N/N_0 \leqq 2.67$$

3.4 Conclusion

In the rate-monotonic-next-fit scheduling algorithm, while assigning the next job to a processor, only the last processor used is checked to see whether or not this job can be assigned to that processor, even though this job may possibly be feasibly assigned to one of the processors used earlier. If earlier processors used are also considered for assignment of the next job, it may be possible to reduce the number of processors used. This approach is considered in the next chapter.

CHAPTER 4

RATE-MONOTONIC-FIRST-FIT SCHEDULING ALGORITHM

4.1 Introduction

According to the rate-monotonic-next-fit scheduling algorithm considered in chapter 3, we only attempt to assign a job to the processor currently under examination, ignoring the possibility of scheduling the job on a processor examined earlier. Intuitively, before assigning a job to a new processor, one might wish to consider whether the job can be feasibly scheduled on any one of the processors examined earlier. The rate-monotonic-first-fit scheduling algorithm, introduced in this chapter, is an attempt in this direction. According to this algorithm, a job is not assigned to the j^{th} processor until it is determined that it cannot be feasibly scheduled on any one of the (j-1) processors examined earlier. Intuitively, therefore, one would expect that the rate-monotonic-first-fit scheduling algorithm will perform better than the rate-monotonic-next-fit scheduling algorithm.

4.2 Rate-Monotonic-First-Fit Scheduling Algorithm

According to the rate-monotonic-first-fit scheduling algorithm, the jobs are first arranged in descending order of their request rates. (For convenience, the jobs are renumbered as J_1, J_2, ..., J_m.) The assignment procedure is as follows:

Step 1: Set i = 1.

Step 2: (a) Set $j = 1$.

(b) If job J_i together with the jobs already assigned to processor P_j can be feasibly scheduled on processor P_j according to the rate-monotonic scheduling algorithm, assign J_i to P_j, and go to step 3.

(c) Else set $j = j + 1$ and go to step 2(b).

Step 3: If all jobs have been assigned then stop; else set $i = i + 1$ and go to step 2.

The largest index j used in the above algorithm is the number of processors required to schedule the given set of jobs according to this algorithm.

We illustrate how this algorithm works by an example.

Example: Let (1, 2), (1, 3), (1, 4), (1.9, 5), (2, 6), (2.5, 7), (3, 8), (3, 9), (3.7, 10), (1, 11), (4, 12), (2, 13), (2, 14), (6, 18), (5, 20) and (8, 24) be a set of jobs. According to the rate-monotonic-first-fit algorithm, jobs (1, 2), (1, 3) and (1, 11) will be assigned to processor P_1; jobs (1, 4), (1.9, 5) and (2, 13) will be assigned to processor P_2; jobs (2, 6), (2.5, 7) and (2, 14) to processor P_3; jobs (3, 8) and (3, 9) to processor P_4; jobs (3.7, 10) and (4, 12) to processor P_5; jobs (6, 18) and (5, 20) to processor P_6; and job (8, 24) will be assigned to processor P_7. Thus a total of 7 processors will be used.

The following partitioning will, however, require only 5 processors:

Group 1: Jobs (1, 2), (3, 8) and (1, 11).

<u>Group 2</u>: Jobs (1, 3), (3, 9) and (6, 18).

<u>Group 3</u>: Jobs (1, 4), (2.5, 7), (2, 13) and (2, 14).

<u>Group 4</u>: Jobs (1.9, 5), (3.7, 10) and (5, 20).

<u>Group 5</u>: Jobs (2, 6), (4, 12) and (8, 24).

Since the utilization factor of this set of jobs is 4.97, this is the minimum number of processors required.

We now proceed to analyze the behavior of the rate-monotonic-first-fit scheduling algorithm. We establish first the following result.

<u>Theorem 5</u>: If a set of three jobs cannot be feasibly scheduled on two processors according to the rate-monotonic-first-fit scheduling algorithm, then the utilization factor of the set of jobs is greater than $3/(1 + 2^{1/3})$.

<u>Proof</u>: Let $\{J_i \equiv (C_i, T_i)\}_{i=1}^{3}$ be a set of three jobs with $T_1 \leq T_2 \leq T_3$. Let the utilization factor of this set of jobs be less than or equal to $3/(1 + 2^{1/3})$. Assume further that this set cannot be feasibly scheduled on two processors according to the rate-monotonic-first-fit scheduling algorithm. This implies that no two of these three jobs can be feasibly scheduled on a single processor according to the rate-monotonic scheduling algorithm. This in turn implies that the utilization factor of any set of two jobs is greater than $2(2^{1/2} - 1)$ (Theorem A, chapter 2). Since $2(2^{1/2} - 1) + 1/2 > 3/(1 + 2^{1/3})$, it follows that the utilization factor of each of the three jobs is less than 1/2. Therefore,

$$2C_i < T_i, \qquad i = 1, 2, 3. \tag{4.1}$$

Further, it is claimed that $T_2 < 3T_1$. According to Theorem B (chapter 2), a set of two jobs (C_1, T_1) and (C_2, T_2), with $\lfloor T_2/T_1 \rfloor = m$ can be feasibly scheduled on a single processor by the rate-monotonic scheduling algorithm if $C_1/T_1 + C_2/T_2 \leqq 2(\sqrt{m(m+1)} - m)$. Therefore, if $T_2 \geqq 3T_1$ (implying $\lfloor T_2/T_1 \rfloor \geqq 3$), we must have:

$$C_1/T_1 + C_2/T_2 > 2(\sqrt{12} - 3)$$

$$C_1/T_1 + C_3/T_3 > 2(\sqrt{12} - 3)$$

$$C_2/T_2 + C_3/T_3 > 2(\sqrt{2} - 1)$$

This would give

$$C_1/T_1 + C_2/T_2 + C_3/T_3 > 2(\sqrt{12} - 3) + (\sqrt{2} - 1)$$

$$> 3/(1 + 2^{1/3}).$$

This contradicts the assumption that the utilization factor of the given set of jobs is less than or equal to $3/(1 + 2^{1/3})$. This establishes our claim that

$$T_2 < 3T_1 \tag{4.2}$$

With J_1 assigned to processor P_1, and J_2 assigned to processor P_2, let S denote the set of all those jobs $J \equiv (C, T)$, such that neither J_1 and J can be feasibly scheduled on processor P_1 nor J_2 and J can be feasibly scheduled on processor P_2 according to the rate-monotonic scheduling algorithm. Let us find a job $J \equiv (C, T)$ belonging to the set S, whose utilization factor is minimum among all jobs in S. Let $\lfloor T/T_1 \rfloor = k_1$ and $\lfloor T/T_2 \rfloor = k_2$.

For a given T, let C' and C" be such that the set of jobs (C_1, T_1) and (C', T) fully utilizes the processor P_1, and jobs (C_2, T_2)

and (C'', T) fully utilize the processor P_2. Then

$$C' = \begin{cases} k_1(T_1 - C_1), & \text{if } k_1T_1 \leqq T \leqq k_1T_1 + C_1 \\ T - (k_1 + 1)C_1, & \text{if } k_1T_1 + C_1 \leqq T \leqq (k_1 + 1)\,T_1 \end{cases} \tag{4.3}$$

$$C'' = \begin{cases} k_2(T_2 - C_2), & \text{if } k_2T_2 \leqq T \leqq k_2T_2 + C_2 \\ T - (k_2 + 1)C_2, & \text{if } k_2T_2 + C_2 \leqq T \leqq (k_2 + 1)T_2 \end{cases} \tag{4.4}$$

If $C >$ max (C', C''), then the job (C, T) belongs to the set S.

Let $f_1(T) = C'/T$ and $f_2(T) = C''/T$, where C' and C'' are given by (4.3) and (4.4) respectively.

Let $f(T) = \max(f_1(T), f_2(T))$.

For a given T, f(T) represents the maximum utilization factor that a job can have so that it can be feasibly scheduled either on P_1 or on P_2. The absolute minimum of f(T) is what we want to determine.

We observe that the function $f_1(T)$ decreases monotonically in the interval $[k_1T_1, k_1T_1 + C_1]$ and increases monotonically in the interval $[k_1T_1 + C_1, (k_1 + 1)T_1]$. Similarly the function $f_2(T)$ decreases monotonically in the interval $[k_2T_2, k_2T_2 + C_2]$ and increases monotonically in the interval $[k_2T_2 + C_2, (k_2 + 1)T_2]$.

In figure 4.1, curve AA' is the graph of the function $f_1(T)$ and curve BB' is that of $f_2(T)$. The heavily marked portions of the two curves represent the function f(T).

It is claimed that the decreasing segments of the two curves $f_1(T)$ and $f_2(T)$ are either identical or have no point in common. This can be seen as follows:

Suppose in an interval $[a, b]$ both $f_1(T)$ and $f_2(T)$ are decreasing. Let $\lfloor a/T_1 \rfloor = k'$ and $\lfloor a/T_2 \rfloor = k''$. For all $x \in [a, b]$

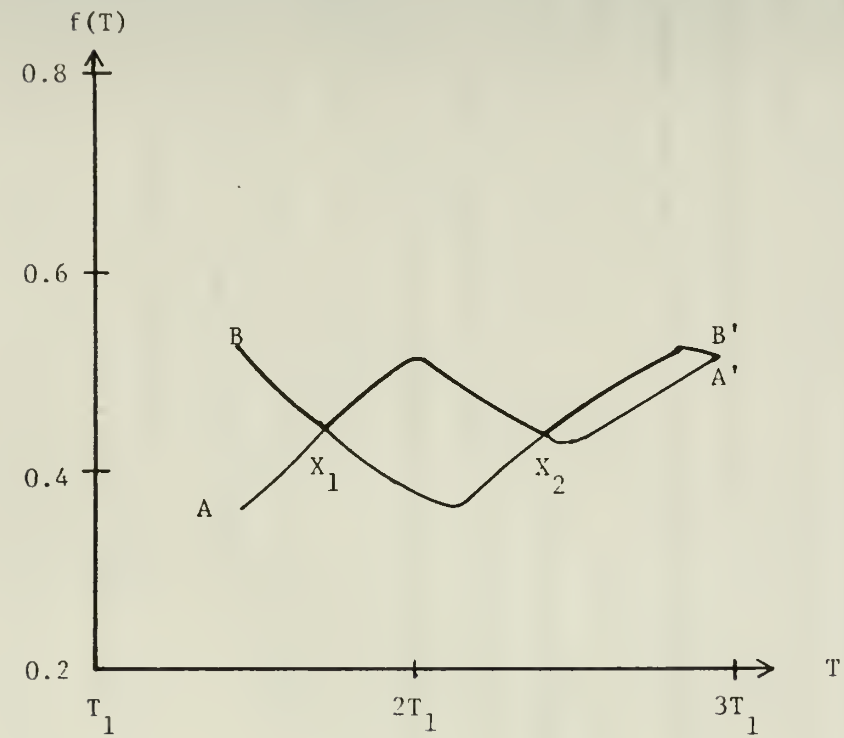

Figure 4.1 Graph representing the maximum utilization factor of a job that fully utilizes processor P_1 along with job J_1, and processor P_2 along with job J_2.

$f_1(x) - f_2(x) = [k'(T_1 - C_1) - k''(T_2 - C_2)]/x$ is less than, equal to or greater than zero, according as $k'(T_1 - C_1) - k''(T_2 - C_2)$ is, or according as $f_1(a) - f_2(a)$ is. Since both $f_1(T)$ and $f_2(T)$ are continuous positive-valued functions in the interval [a, b] and $f_1(T) - f_2(T)$ has the same sign throughout the interval [a, b], they do not cross each other at any point in the interval.

We can similarly show that the increasing segments of the two curves $f_1(T)$ and $f_2(T)$ are either identical or have no point in common.

Thus at a point of intersection of the two curves $f_1(T)$ and $f_2(T)$, one of the functions is increasing and the other is decreasing. Also note that the points of minima of the function $f(T)$ are among the points of minima of the functions $f_1(T)$ and $f_2(T)$ and the points of intersection of the two curves.

We consider the following cases:

<u>Case 1</u>: $\quad T_2 \leqq T_1 + C_1 \qquad (4.5)$

Since J_1 and J_2 cannot be feasibly scheduled on P_1, we have

$$C_2 > T_1 - C_1 \qquad (4.6)$$

In this case, we must have $T_3 < 3T_1$, because if $T_3 \geqq 3T_1$, then $T_3 \geqq 2T_2$, and, therefore, according to Theorem B (chapter 2),

$$C_1/T_1 + C_2/T_2 > 2(\sqrt{2} - 1)$$

$$C_2/T_2 + C_3/T_3 > 2(\sqrt{6} - 2)$$

$$C_1/T_1 + C_3/T_3 > 2(\sqrt{12} - 3)$$

which gives $C_1/T_1 + C_2/T_2 + C_3/T_3 > 3/(1 + 2^{1/3})$, contradicting the assumption that the utilization factor of the given set of jobs is less than or equal to $3/(1 + 2^{1/3})$.

We now show that in this case the two curves $f_1(T)$ and $f_2(T)$ do intersect in the interval $[T_1 + C_1, 2T_1]$.

When $T = T_1 + C_1$,

$$f_1(T) = (T_1 - C_1)/T$$

$$f_2(T) = (T_2 - C_2)/T$$

Since $T_2 > 2C_2$ and $C_2 > T_1 - C_1$, it follows that

$$f_2(T) > f_1(T) \text{ at } T = T_1 + C_1 \tag{4.7}$$

When $T = 2T_1$,

$$f_1(T) = 2(T_1 - C_1)/2T_1$$

and

$$f_2(T) = \begin{cases} (T_2 - C_2)/2T_1, & \text{if } 2T_1 \leqq T_2 + C_2 \\ 2(T_1 - C_2)/2T_1, & \text{if } T_2 + C_2 \leqq 2T_1 \end{cases}$$

Since $C_1 < T_1 - C_1 < C_2$ and $T_2 < T_1 + C_1$, we get

$$f_2(T) < f_1(T) \text{ at } T = 2T_1 \tag{4.8}$$

Since $f_1(T)$ and $f_2(T)$ are single-valued continuous functions, it follows from (4.7) and (4.8), that the two curves must intersect at a point somewhere in the interval $[T_1 + C_1, 2T_1]$. Let X_1 be this point of intersection (figure 4.1).

At X_1, T is given by the equation:

$$T - 2C_1 = T_2 - C_2$$

i.e., $$T = T_2 - C_2 + 2C_1 \tag{4.9}$$

and $$C = T_2 - C_2 \tag{4.10}$$

From (4.1), (4.5) and (4.6), it follows that $(T_1 - C_1) < C_2 < (T_1 + C_1)/2$. Once C_1, T_1 and T_2 are fixed as above, C_2 can have any value in the interval $((T_1 - C_1), \frac{1}{2}(T_1 + C_1))$. If we choose C_2, C_3 and T_3 such that $C_1/T_1 + C_2/T_2 + C_3/T_3$ is minimum over

all possible choices of C_2, C_3 and T_3, and C_3 and T_3 are such that once C_2 is chosen, the point (C_3, T_3) is the minimum of the curve $f(T)$, then a set of jobs having utilization factor less than or equal to $C_1/T_1 + C_2/T_2 + C_3/T_3$ and satisfying the conditions (4.1), (4.2), (4.5) and (4.6) can be feasibly scheduled on two processors according to the rate-monotonic-first-fit scheduling algorithm. We claim that these requirements will be satisfied, if we choose

$$C_2 = T_1 - C_1$$

$$C_3 = T_2 - C_2 = T_2 - T_1 + C_1$$

$$T_3 = T_2 - C_2 + 2C_1 = T_2 - T_1 + 3C_1$$

and then

$$U = \frac{C_1}{T_1} + \frac{T_1 - C_1}{T_2} + \frac{T_2 - T_1 + C_1}{T_2 - T_1 + 3C_1} .$$

Let $C_2' = T_1 - C_1 + \Delta$, with $\Delta > 0$.

To find the corresponding pair (C_3', T_3'), we have to find the corresponding point where the function $f(T)$ attains its minimum. Since this point is attained at one of the points of intersection of the two curves $f_1(T)$ or $f_2(T)$, or one of the points of minima of these two curves, we examine all such possible points.

At the point of intersection X_1 in the interval $[T_1 + C_1, 2T_1]$,

$$\begin{aligned} T_3' &= T_2 - C_2' + 2C_1 \\ &= T_2 - T_1 + 3C_1 - \Delta \\ &= T_3 - \Delta \end{aligned}$$

and,

$$\begin{aligned} C_3' &= T_2 - C_2' \\ &= T_2 - T_1 + C_1 - \Delta \\ &= C_3 - \Delta \end{aligned}$$

Let

$$U' = C_1/T_1 + C_2'/T_2 + C_3'/T_3'$$

Then

$$U' - U = \Delta/T_2 + (C_3 - \Delta)/(T_3 - \Delta) - C_3/T_3$$

$$= \Delta/T_2 - \Delta(T_3 - C_3)/T_3(T_3 - \Delta)$$

$$= \Delta[T_3(T_3 - \Delta) - T_2(T_3 - C_3)]/T_2T_3(T_3 - \Delta).$$

Since $C_3 = T_2 - C_2 > \frac{1}{2}T_2 > C_2' = C_2 + \Delta > \Delta$

and $T_3 \geqq T_2$,

we have, $U' - U > 0$.

The next point of intersection is either in the interval $[2T_1, 2T_1 + C_1]$ or there is no other point of intersection upto $3T_1$.

Suppose there is a point X_2 in the interval $[2T_1, 2T_1 + C_1]$ where the two curves $f_1(T)$ and $f_2(T)$ intersect.

At X_2,

$$C_3' = 2(T_1 - C_1) = 2C_2$$

and $$T_3' = 2(T_1 - C_1 + C_2') = 2(2C_2 + \Delta)$$

$$U' = C_1/T_1 + (C_2 + \Delta)/T_2 + 2C_2/2(2C_2 + \Delta)$$

$$U' - U = C_1/T_1 + C_2/(2C_2 + \Delta) - (T_2 - C_2)/(T_2 - C_2 + 2C_1)$$

$$= \frac{\Delta}{T_2} + \frac{C_2(2C_1 + C_2 - T_2) - \Delta(T_2 - C_2)}{(2C_2 - \Delta)(T_2 - C_2 + 2C_1)}$$

$$= \frac{\Delta\{(2C_2 + \Delta)(T_2 - C_2 + 2C_1) - T_2(T_2 - C_2)\}}{T_2(2C_2 + \Delta)(T_2 - C_2 + 2C_1)}$$

$$+ \frac{C_2(2C_1 + C_2 - T_2)}{(2C_2 + \Delta)(T_2 - C_2 + 2C_1)}$$

$$U' - U = \frac{\Delta\ \{T_2(3C_2 - T_2) + \Delta T_2 + (2C_1 - C_2)(2C_2 + \Delta)\}}{T_2(2C_2 + \Delta)(T_2 - C_2 + 2C_1)}$$

$$+ \frac{C_2(2C_1 + C_2 - T_2)}{(2C_2 + \Delta)(T_2 - C_2 + 2C_1)} \qquad (4.11)$$

Since $T_2 \leqq T_1 + C_1$ and $C_2 = T_1 - C_1$, we have

$$2C_1 + C_2 - T_2 \geqq 2C_1 + T_1 - C_1 - T_1 - C_1 = 0 \qquad (4.12)$$

Also $3T_1 - T_2 \geqq 3T_1 - (T_1 + C_1) = 2T_1 - C_1 \geqq 4C_1 - C_1 = 3C_1$

Therefore, $3(T_1 - C_1) \geqq T_2$

or $3C_2 \geqq T_2$ (4.13)

Further, since $T_1 - C_1 = C_2 < \frac{1}{2}(T_1 + C_1)$

$$T_1 < 3C_1$$

Hence, $C_2 = T_1 - C_1 < 3C_1 - C_1 = 2C_1$ (4.14)

From (4.11), (4.12), (4.13) and (4.14), we see that

$$U' - U > 0.$$

The other possibility is that there is no point of intersection of $f_1(T)$ and $f_2(T)$ in the interval $[2T_1, 2T_1 + C_1]$ (See figure 4.2).

Since at $T = 2T_1$, $f_2(T) \leqq f_1(T)$, (from 4.8), in this case, we should also have $f_2(T) \leqq f_1(T)$, when $T = 2T_1 + C_1$.

Therefore, $(T - 2C_2')/T \leqq 2(T_1 - C_1)/T$, where $T = 2T_1 + C_1$

Hence $2T_1 + C_1 - 2C_2' \leqq 2T_1 - 2C_1$

or, $3C_1 \leqq 2C_2'$ (4.15)

In this case, the minimum of $f(T)$ can occur at the minimum of the curve $f_1(T)$ in the interval $[2T_1, 3T_1]$ which is at point $T = 2T_1 + C_1$.

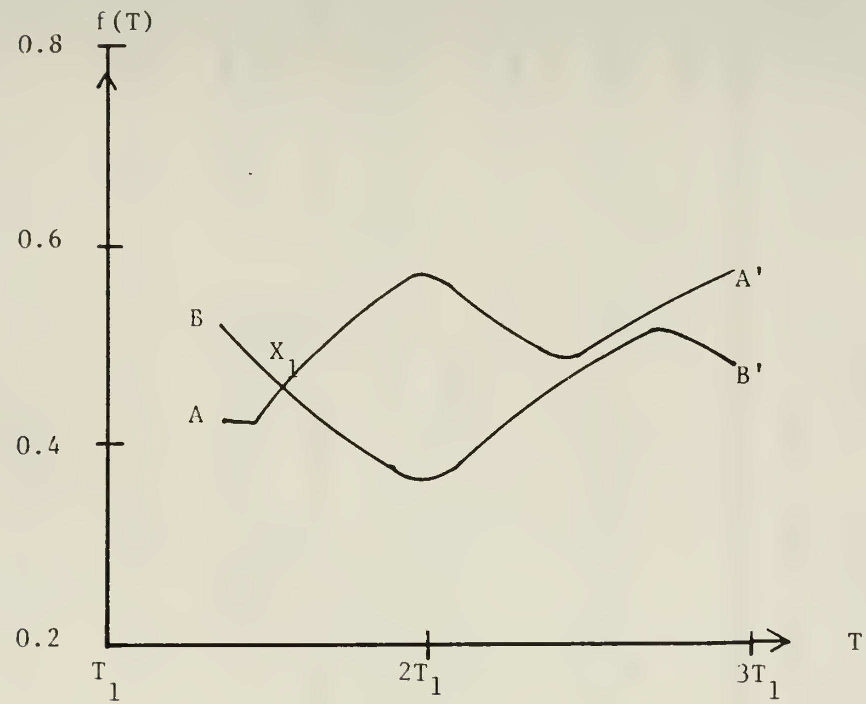

Figure 4.2 Graph representing the maximum utilization factor of a job that fully utilizes processor P_1 along with job J_1^1, and processor P_2 along with job J_2^1.

In that case, $C_3' = 2(T_1 - C_1) = 2C_2$

and $T_3' = 2T_1 + C_1$

Then,

$$U' = C_1/T_1 + (C_2 + \Delta)/T_2 + 2C_2/(2T_1 + C_1)$$

$$U' - U = \Delta/T_2 + 2C_2/(2T_1 + C_1) - (T_2 - C_2)/(T_2 - C_2 + 2C_1)$$

$$= \frac{\Delta}{T_2} + \frac{2(T_1 - C_1)}{2T_1 + C_1} - \frac{T_2 - T_1 + C_1}{T_2 - T_1 + 3C_1}$$

$$= \frac{\Delta}{T_2} + \frac{7C_1T_1 - 3C_1T_2 - 7C_1^2}{(2T_1 + C_1)(T_2 - T_1 + 3C_1)} \qquad (4.16)$$

From (4.15), $2C_2' \geqq 3C_1$

or, $2(T_1 - C_1 + \Delta) \geqq 3C_1$

or, $2T_1 - 5C_1 \geqq -2\Delta$ (4.17)

Also, since $T_2 \leqq T_1 + C_1$

we have,

$$7C_1T_1 - 3C_1T_2 - 7C_1^2 \geqq 7C_1T_1 + 3C_1(-T_1 - C_1) - 7C_1^2$$

$$= 4C_1T_1 - 10C_1^2$$

$$= 2C_1(2T_1 - 5C_1)$$

Using (4.17), we get

$$7C_1T_1 - 3C_1T_2 - 7C_1^2 \geqq -4 \cdot \Delta C_1 \qquad (4.18)$$

Therefore, from (4.16) and (4.18),

$$U' - U \geqq \Delta/T_2 - 4\Delta C_1/(2T_1 + C_1)(T_2 - T_1 + 3C_1)$$

$$= \frac{\Delta\ \{T_2(2T_1 - 3C_1) + (5C_1T_1 + 3C_1^2 - 2T_1^2)\}}{T_2(2T_1 + C_1)(T_2 - T_1 + 3C_1)}$$

$$\geqq \frac{\Delta\ \{T_1(2T_1 - 3C_1) + 5C_1T_1 + 3C_1^2 - 2T_1^2\}}{T_2(2T_1 + C_1)(T_2 - T_1 + 3C_1)}$$

$$U' - U \geqq \Delta\{\ 2C_1T_1 + 3C_1^2\ \}/T_2(2T_1 + C_1)(T_2 - T_1 + 3C_1)$$
$$> 0.$$

Another point that is a potential minimum of f(T) is the point of intersection of the two curves $f_1(T)$ and $f_2(T)$ in the interval $[2T_1 + C_1,\ 3T_1]$ provided it exists. This point is determined by the equation:

$$(T_3' - 3C_1)/T_3' = 2(T_2 - C_2')/T_3'$$

Thus in this case,

$$T_3' = 2(T_2 - C_3') + 3C_1,$$

and then,

$$C_3' = 2(T_2 - C_2')$$

$$U' = \frac{C_1}{T_1} + \frac{(C_2 + \Delta)}{T_2} + \frac{2(T_2 - C_2 - \Delta)}{2(T_2 - C_2 - \Delta\ ) + 3C_1}$$

The two curves $f_1(T)$ and $f_2(T)$ can intersect in the interval $[2T_1 + C_1,\ 3T_1]$ only if there is also a point of intersection of the two curves in the interval $[\ 2T_1,\ 2T_1 + C_1\ ]$ (see figure 4.3). At that point of intersection,

$$T_3'' = T_2 - C_2' + 2C_1$$

and $\quad C_3'' = T_2 - C_2,$

and the corresponding utilization factor is,

$$U'' = C_1/T_1 + (C_2 + \Delta)/T_2 + (T_2 - C_2 - \Delta\ )/(T_2 - C_2 - \Delta + 2C_1)$$

It has already been shown that $U'' > U$.

Now,

$$U' = C_1/T_1 + (C_2 + \Delta)/T_2 + 2(T_2 - C_2 - \Delta)/\{\ 2(T_2 - C_2 - \Delta) + 3C_1\}$$

$$= C_1/T_1 + (C_2 + \Delta)/T_2 + (T_2 - C_2 - \Delta)/\{T_2 - C_2 - \Delta + 3C_1/2\} > U''.$$

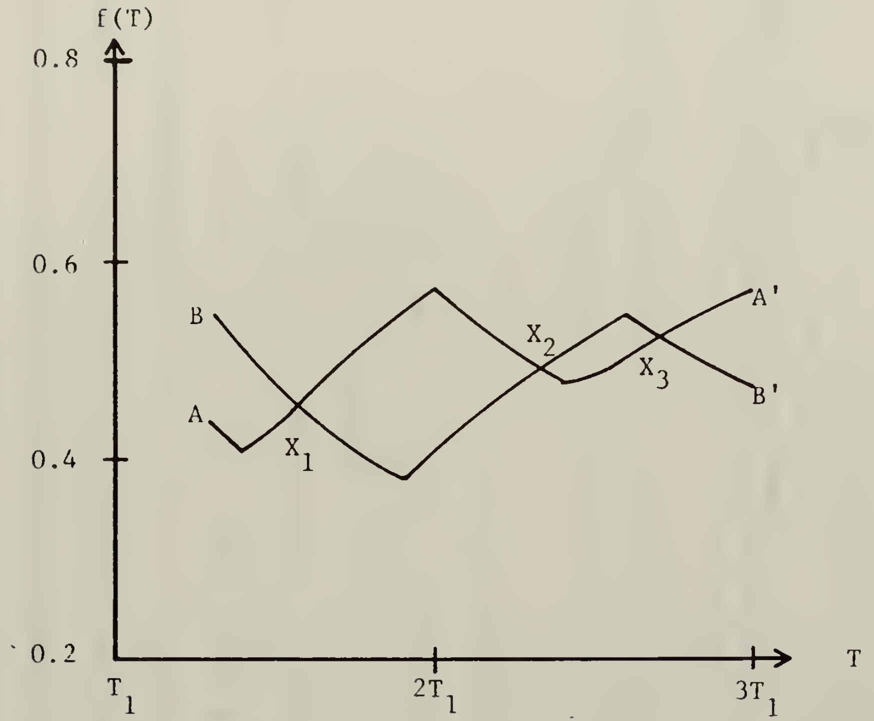

Figure 4.3 Graph representing the maximum utilization factor of a job that fully utilizes processor P_1 along with job J_1, and processor P_2 along with job J_2.

Hence, $U' > U$.

Thus for fixed C_1, T_1 and T_2, with $T_2 \leqq T_1 + C_1$, the utilization factor

$$U = C_1/T_1 + (T_1 - C_1)/T_2 + (T_2 - T_1 + C_1)/(T_2 - T_1 + 3C_1)$$

is such that for any choice of C_2 in the range $((T_1 - C_1), (T_1 + C_1)/2))$ and any C_3, T_3 such that $C_1/T_1 + C_2/T_2 + C_3/T_3$ is less than or equal to U, the set of jobs (C_1, T_1), (C_2, T_2), and (C_3, T_3) can be feasibly scheduled on two processors according to the rate-monotonic-first-fit scheduling algorithm.

Since $T_2 > 2C_2 = 2(T_1 - C_1)$, we let $T_2 = 2(T_1 - C_1) + k$, where k is greater than or equal to zero. Then,

$$U = C_1/T_1 + (T_1 - C_1)/\{2(T_1 - C_1) + k\} + (T_1 - C_1 + k)/(T_1 + C_1 + k).$$

Writing x for C_1/T_1, and y for k/T_1, we can write

$$U = x + \frac{1 - x}{2(1 - x) + y} + \frac{1 - x + y}{1 + x + y} \tag{4.19}$$

To determine the minimum value of U over all possible sets of three jobs, satisfying the conditions of this case, the expression in (4.19) is to be minimized. Setting partial derivatives of U with respect to x and y to zero, we have:

$$\frac{\partial U}{\partial x} = 1 - \frac{y}{\{2(1 - x) + y\}^2} - \frac{2(1 + y)}{(1 + x + y)^2} = 0$$

$$\frac{\partial U}{\partial y} = - \frac{(1 - x)}{\{2(1 - x) + y\}^2} + \frac{2x}{(1 + x + y)^2} = 0$$

Solving these two equations for x and y, we get:

$$x = 1/(1 + 2^{1/3})$$

$$y = (2^{2/3} - 2^{1/3})/2,$$

which gives $U = 3/(1 + 2^{1/3})$.

Thus, we conclude in this case that if three jobs cannot be feasibly scheduled on two processors according to the rate-monotonic-first-fit scheduling algorithm, their utilization factor must be greater than $3/(1 + 2^{1/3})$.

Case 2: $T_1 + C_1 \leqq T_2 \leqq 2T_1$ (4.20)

Since J_1 and J_2 cannot be feasibly scheduled on a single processor by the rate-monotonic scheduling algorithm, we must have:

$$C_2 > T_2 - 2C_1 > T_1 - C_1 \qquad (4.21)$$

Also, we claim that in this case $T_3 \leqq 4T_1$. Because otherwise by Theorem B (chapter 2),

$$C_1/T_1 + C_2/T_2 > 2(\sqrt{2} - 1)$$

$$C_1/T_1 + C_3/T_3 > 2(\sqrt{20} - 4)$$

$$C_2/T_2 + C_3/T_3 > 2(\sqrt{6} - 2)$$

which would give

$C_1/T_1 + C_2/T_2 + C_3/T_3 > 3/(1 + 2^{1/3})$, contrary to our assumption.

The possible configurations for the graphs of $f_1(T)$ and $f_2(T)$, and consequently that of $f(T)$, are shown in figures 4.4, 4.5, 4.6, 4.7, 4.8 and 4.9.

As shown in case 1, we can prove that in this case also the two curves $f_1(T)$ and $f_2(T)$ must intersect in the interval $[T_1 + C_1, 2T_1]$. However, there might or might not be other points of intersection in the interval $[2T_1, 4T_1]$.

It will be shown below that for fixed C_1, T_1, C_2, and T_2, the absolute minimum of $f(T)$ occurs -

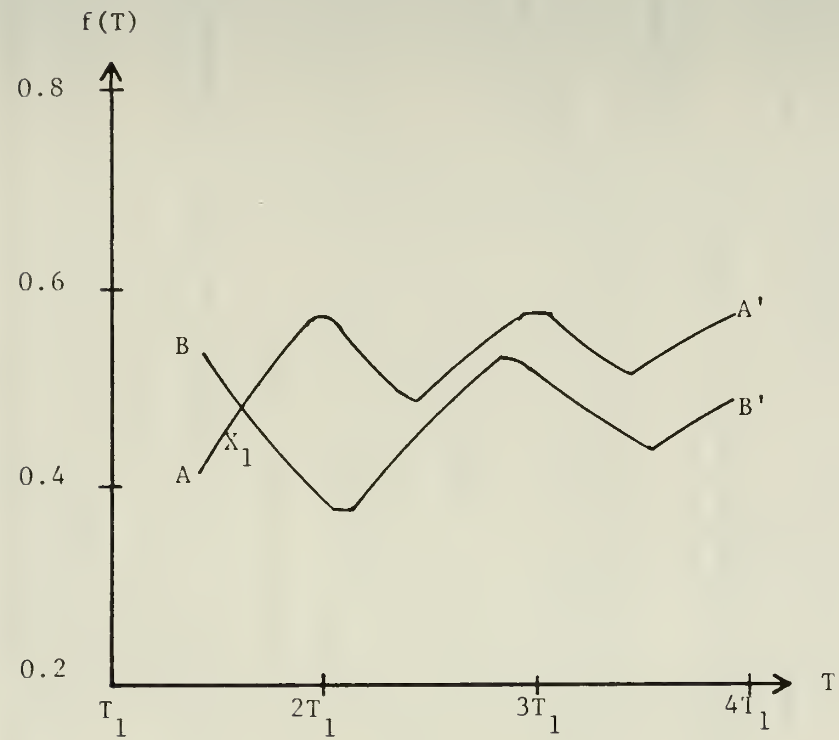

Figure 4.4 Graph representing the maximum utilization factor of a job that fully utilizes processor P_1 along with job J_1, and processor P_2 along with job J_2.

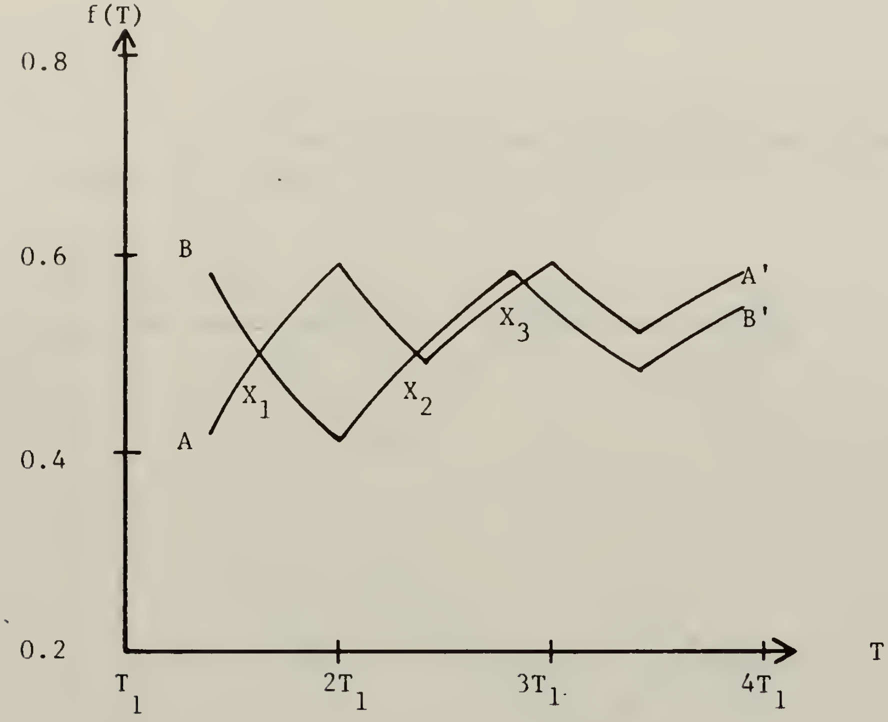

Figure 4.5 Graph representing the maximum utilization factor of a job that fully utilizes processor P_1 along with job J_1, and processor P_2 along with job J_2.

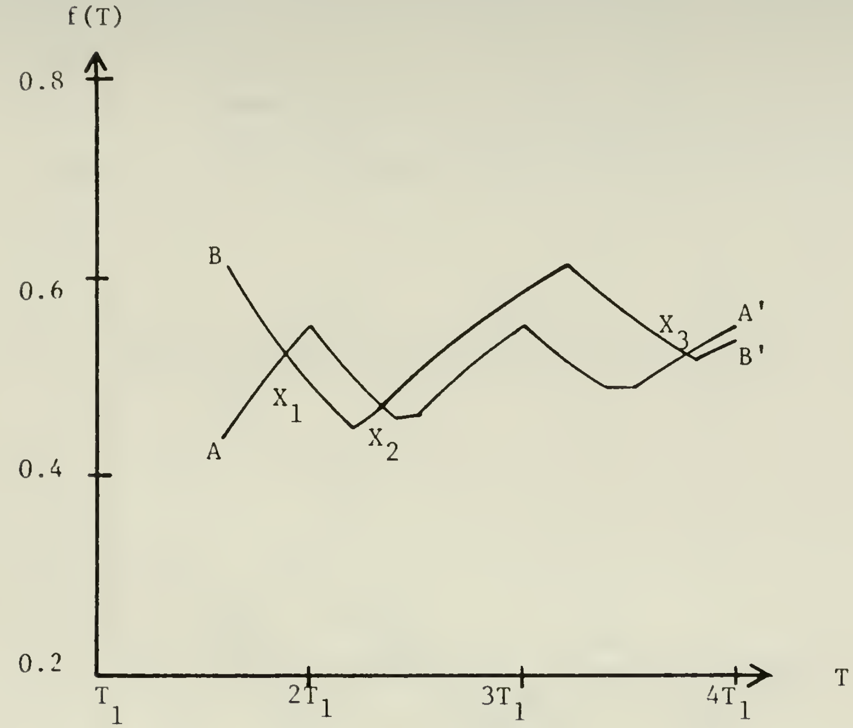

Figure 4.6 Graph representing the maximum utilization factor of a job that fully utilizes processor P_1 along with job J_1, and processor P_2 along with job J_2.

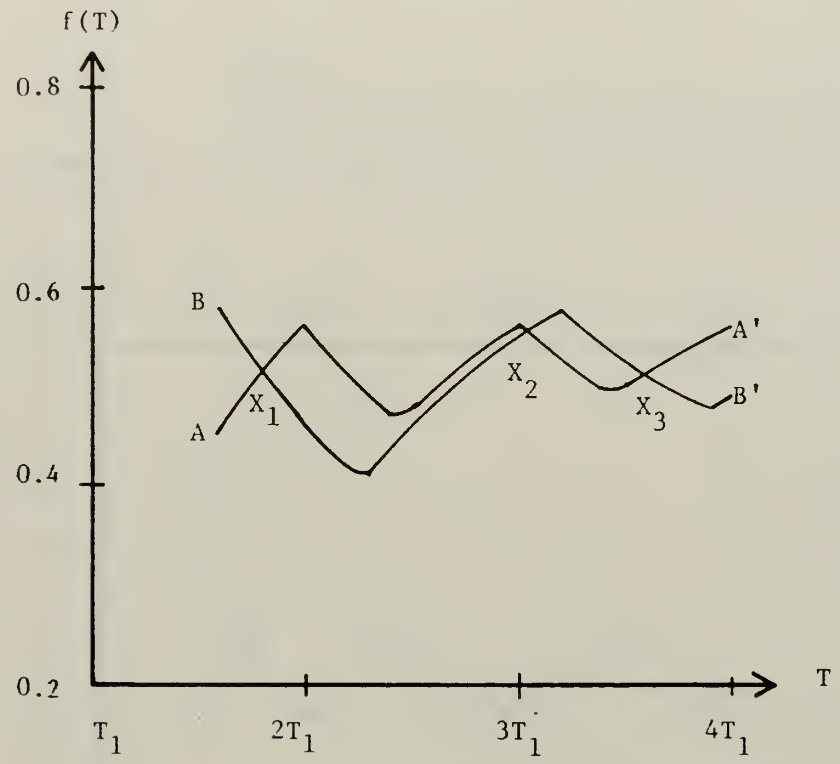

Figure 4.7 Graph representing the maximum utilization factor of a job that fully utilizes processor P_1 along with job J_1, and processor P_2 along with job J_2.

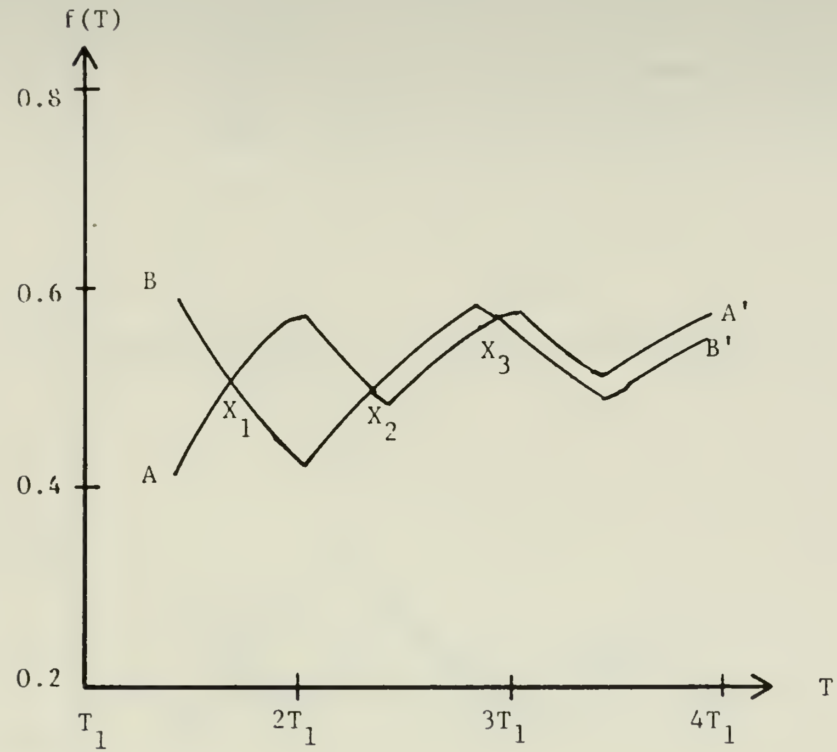

Figure 4.8 Graph representing the maximum utilization factor of a job that fully utilizes processor P_1 along with job J_1, and processor P_2 along with job J_2.

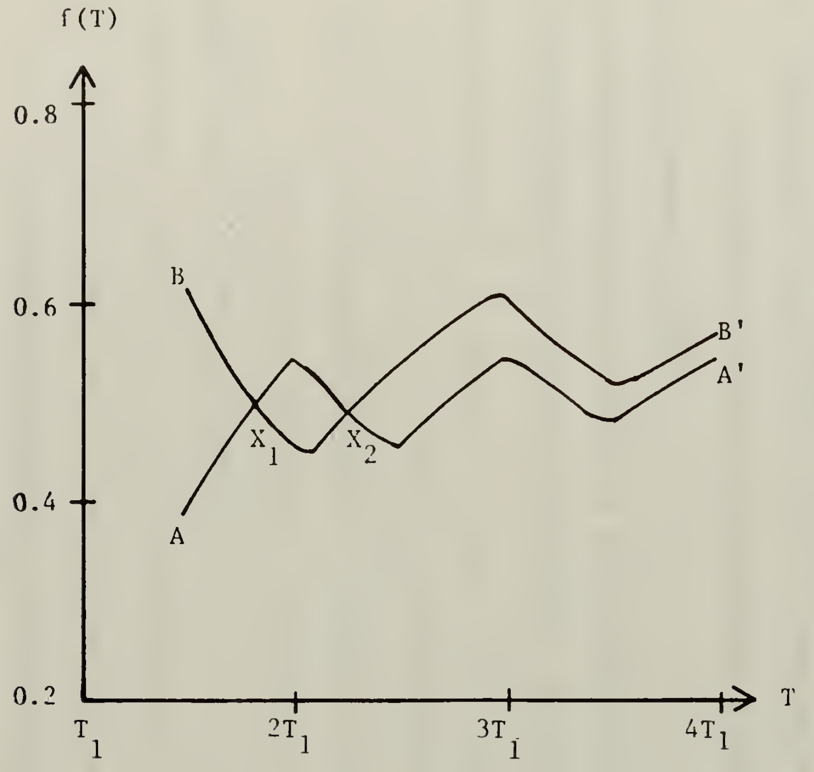

Figure 4.9 Graph representing the maximum utilization factor of a job that fully utilizes processor P_1 along with job J_1, and processor P_2 along with job J_2.

either (i) at the point of intersection of the two curves $f_1(T)$ and $f_2(T)$ in the interval $[T_1 + C_1, 2T_1]$,

or (ii) at the point of intersection of the two curves in the interval $[2T_1, 2T_1 + C_1]$, if there is one, or if there is no point of intersection in this interval, at the point $2T_1 + C_1$, which is one of the points of minima of $f_1(T)$.

If there is no point of intersection in the interval $[2T_1, 2T_1 + C_1]$, then there is also no point of intersection in the interval $[2T_1 + C_1, 3T_1]$, and $f_2(T) \leqq f_1(T)$ for $T \in [2T_1, 3T_1]$. Thus, in the interval $[2T_1, 3T_1]$,

$$f(T) = \max \{f_1(T), f_2(T)\} = f_1(T).$$

Since $f_1(T)$ is minimum at $T = 2T_1 + C_1$ in the interval $[2T_1, \infty)$, and since $f(T) \geqq f_1(T)$ in the interval $[2T_1, \infty)$, the minimum of $f(T)$ in the interval $[2T_1, \infty)$ occurs at $T = 2T_1 + C_1$. Thus in this case, the points where $f(T)$ can have its minimum is either the point of intersection of the two curves in the interval $[T_1 + C_1, 2T_1]$, or the point where $T = 2T_1 + C_1$.

If the two curves do intersect at a point, X_2, say, in the interval $[2T_1, 2T_1 + C_1]$, then we claim that at other points of minima of $f(T)$ in the interval $[2T_1 + C_1, 4T_1]$ the value of the function $f(T)$ will either be greater than that at X_2 or greater than that at X_1. This can be seen as follows.

If the two curves $f_1(T)$ and $f_2(T)$ intersect in the interval $[2T_1 + C_1, 3T_1]$, that point of intersection is given by the equation:

$$(T - 3C_1)/T = 2(T_2 - C_2)/T$$

so that $T = 2(T_2 - C_2) + 3C_1$.

The value of f(T) at this point is, therefore, equal to $2(T_2 - C_2)/\{2(T_2 - C_2) + 3C_1\}$ which is greater than $(T_2 - C_2)/\{(T_2 - C_2) + 2C_1\}$ which is the value of the function f(T) at X_1, the point of intersection of $f_1(T)$ and $f_2(T)$ in the interval $[T_2, 2T_1]$.

Next suppose there is a point of intersection in the time interval $[3T_1, 3T_1 + C_1]$. This point is given by the equation:

$$3(T_1 - C_1)/T = (T - 3C_2)/T$$

which gives $T = 3(T_1 - C_1) + 3C_2$, and then $C = 3(T_1 - C_1)$. Hence, at this point of intersection,

$$\begin{aligned} f(T) &= 3(T_1 - C_1)/\{3(T_1 - C_1) + 3C_2\} \\ &= (T_1 - C_1)/(T_1 - C_1 + C_2) \\ &= f(T) \text{ at } X_2. \end{aligned}$$

Suppose further that there is a point of intersection of the two curves in the interval $[3T_1 + C_1, 4T_1]$. There are two situations under which the two curves can intersect in this interval:

(i) There is a point of intersection in each of the two intervals $[2T_1 + C_1, 3T_1]$ and $[3T_1, 3T_1 + C_1]$;

or (ii) there is no point of intersection in either one of the above two intervals.

In situation (i), the value of the function f(T) at this point of intersection is $3(T_2 - C_2)/\{3(T_2 - C_2) + 4C_1\}$ which is greater than $(T_2 - C_2)/(T_2 - C_2 + 2C_1)$, the value of f(T) at X_1.

In situation (ii), the value of the function f(T) at

at the point of intersection in the interval $[3T_1 + C_1, 4T_1]$ is $2(T_2 - C_2)/\{2(T_2 - C_2) + 4C_1\} = (T_2 - C_2)/(T_2 - C_2 + 2C_1) = f(T)$ at X_1.

Another point that is a potential minimum for f(T) is the point where $T = 3T_1 + C_1$, provided there is a point of intersection in the interval $[2T_1 + C_1, 3T_1]$, but there is no point of intersection in the interval $[3T_1, 4T_1]$, as shown in figure 4.8. This situation can arise in one of the following two cases:

$$\text{Either (i) } 3(T_1 - C_1) > 2(T_2 - C_2) \tag{4.22}$$

$$\text{or (ii) } 3(T_1 - C_1) > 3T_1 + C_1 - 3C_2$$

$$\text{i.e. } 3C_2 > 4C_1 \tag{4.23}$$

In case (i), we claim that f(T) at $T = 3T_1 + C_1$ is greater than f(T) at X_1. This will be so,

if, $3(T_1 - C_1)/(3T_1 + C_1) > (T_2 - C_2)/(T_2 - C_2 + 2C_1)$

or if, $(1 - C_1/T_1)/(1 + C_1/3T_1) > 1/\{1 + 2C_1/(T_2 - C_2)\}$

or if, $(T_1 - C_1)/(T_2 - C_2) > 2/3$

which is true because of (4.22).

In case (ii), we claim that f(T) at $T = 3T_1 + C_1$ is greater than f(T) at X_2. This will be so,

if, $3(T_1 - C_1)/(3T_1 + C_1) > 2(T_1 - C_1)/2(T_1 - C_1 + C_2)$

or if, $3C_2 > 4C_1$,

which is the same condition as (4.23).

There is one more possibility to consider. That is when the two curves intersect in the interval $[2T_1, 2T_1 + C_1]$, there is no other point of intersection in the interval $[2T_1 + C_1, 4T_1]$, and $f_2(T)$ is greater than $f_1(T)$ in the interval $[X_2, 4T_1]$ (figure 4.9). In this case we claim that the value of f(T) at $T = 2T_2 + C_2$ is greater than that of f(T) either at X_1 or at X_2. Here again, there are two cases:

Either (i) $2(T_2 - C_2) > 3(T_1 - C_1)$ (4.24)

or (ii) $2(T_2 - C_2) > 2T_2 + C_2 - 4C_1$

i.e. $4C_1 > 3C_2$ (4.25)

In case (i), we claim that the value of the function $f(T)$ at $T = 2T_2 + C_2$ is greater than that of $f(T)$ at X_2. This will be so,

if, $2(T_2 - C_2)/(2T_2 + C_2) > 2(T_1 - C_1)/2(T_1 - C_1 + C_2)$

or if, $2(T_2 - C_2)/\{2(T_2 - C_2) + 3C_2\} > 1/\{1 + C_2/(T_1 - C_1)\}$

or if, $(T_2 - C_2)/(T_1 - C_1) > 3/2$,

which is true in view of (4.24).

In case (ii), we claim that the value of $f(T)$ at $T = 2T_2 + C_2$ is greater than that of $f(T)$ at X_1. This will be so,

if, $2(T_2 - C_2)/(2T_2 + C_2) > (T_2 - C_2)/(T_2 - C_2 + 2C_1)$

or if, $4C_1 > 3C_2$,

which is true because of (4.25).

Thus, the minimum of $f(T)$ occurs either at X_1 or at one of the following two points:

(a) The point of intersection of $f_1(T)$ and $f_2(T)$ in the interval $[2T_1, 2T_1 + C_1]$, if it exists, or

(b) The point where $T = 2T_1 + C_1$.

As before, for fixed C_1, T_1, T_2, we can choose any value for C_2 in the range $(T_2 - 2C_1, T_1)$, and then choose C_3, T_3 such that the value of $f(T)$ is minimum at T_3. If we choose C_2, C_3, T_3 such that $C_1/T_1 + C_2/T_2 + C_3/T_3$ is minimum over all possible choices of C_2, C_3 and T_3, then a set of jobs having utilization factor less than or equal to $C_1/T_1 + C_2/T_2 + C_3/T_3$ can be feasibly scheduled on two processors according to the rate-monotonic-first-fit scheduling algorithm. We claim that $C_1/T_1 + C_2/T_2 + C_3/T_3$ is minimum, when

$$C_2 = T_2 - 2C_1$$

$$C_3 = 2(T_1 - C_1)$$

and $$T_3 = 2(T_1 - C_1 + T_2)$$

with

$$U = \frac{C_1}{T_1} + \frac{T_2 - 2C_1}{T_2} + \frac{2(T_1 - C_1)}{2(T_1 - C_1 + T_2)}$$

Since $T_1 + C_1 \leqq T_2 \leqq 2T_1$, we can write

$$T_2 = T_1 + C_1 + \Delta \quad \text{where} \quad 0 \leqq \Delta \leqq T_1 - C_1$$

This gives

$$U = \frac{C_1}{T_1} + \frac{T_1 - C_1 + \Delta}{T_2 + C_2 + \Delta} + \frac{T_1 - C_1}{2(T_1 - C_1) + \Delta}$$

subject to the condition that $0 \leqq \Delta \leqq T_1 - C_1$.

Suppose $C_2' = T_2 - 2C_1 + k$

Then at the point of intersection X_1,

$$C_3' = T_2 - C_2' = T_2 - C_2 - k$$

and $$T_3' = T_2 - C_2' + 2C_1 = T_2 - C_2 - k + 2C_1$$

$$U' = C_1/T_1 + (C_2 + k)/T_2 + (T_2 - C_2 - k)/(T_2 - C_2 - k + 2C_1)$$

$$= C_1/T_1 + (C_2 + k)/T_2 + 2C_1/(4C_1 - k)$$

$$U' - U = k/T_2 + 2C_1/(4C_1 + k) - (T_1 - C_1)/\{2(T_1 - C_1) + \Delta\}$$

$$= \frac{k}{T_1 + C_1 + \Delta} + \frac{\Delta/2}{2(T_1 - C_1) + \Delta} + \frac{k/2}{4C_1 - k}$$

$$= \frac{k\{7C_1 - (T_1 + \Delta + 2k)\}}{(T_1 + C_1 + \Delta)(8C_1 - 2k)} + \frac{\Delta/2}{2(T_1 - C_1) + \Delta}$$

Since $2C_2' \leqq T_2$,

$$2(T_1 - C_1 + \Delta + k) \leqq T_1 + C_1 + \Delta$$

or $T_1 + \Delta + 2k \leqq 3C_1 < 7C_1$.

Hence, $U' - U > 0$.

The other point where U' can be minimum is either the point of intersection of the two curves $f_1(T)$ and $f_2(T)$ in the interval $[2T_1, 2T_1 + C_1]$ if it exists, or else when $T_3' = 2T_1 + C_1$.

If there is no point of intersection in the interval $[2T_1, 2T_1 + C_1]$ then,

$$f_2(2T_1 + C_1) \leqq f_1(2T_1 + C_1)$$

i.e., $$2T_1 + C_1 - 2C_2' \leqq 2(T_1 - C_1)$$

or, $$3C_1 - 2\Delta - 2k \leqq 2(T_1 - C_1)$$

$$-2k \leqq 2T_1 - 5C_1 + 2\Delta \qquad (4.26)$$

At $T_3' = 2T_1 + C_1$, $\quad C_3' = 2(T_1 - C_1)$

Hence,

$$U' = C_1/T_1 + (C_2 + k)/T_2 + 2(T_1 - C_1)/(2T_1 + C_1)$$

$$U' - U = k/T_2 + 2(T_1 - C_1)/(2T_1 + C_1) - (T_1 - C_1)/\{2(T_1 - C_1) + \Delta\}$$

$$= \frac{k}{T_1 + C_1 + \Delta} + \frac{(T_1 - C_1)(2T_1 - 5C_1 + 2\Delta)}{(2T_1 + C_1)(2T_1 - 2C_1 + \Delta)}$$

Using (4.26), we obtain

$$U' - U \geqq k/(T_1 + C_1 + \Delta) - 2k(T_1 - C_1)/(2T_1 + C_1)(2T_1 - 2C_1 + \Delta)$$

$$= k\{2T_1(T_1 - C_1) + 3C_1)\}/(T_1 + C_1 + \Delta)(2T_1 + C_1)(2T_1 - 2C_1 + \Delta)$$

$$> 0.$$

In case the two curves $f_1(T)$ and $f_2(T)$ intersect in the interval $[2T_1 \,,\; 2T_1 + C_1]$, say at X_2, then,

$$f_2(2T_1 + C_1) \geqq f_1(2T_1 + C_1)$$

$$\text{i.e.,} \quad 2T_1 + C_1 - 2C_2' \geqq 2(T_1 - C_1)$$

$$\text{or,} \quad 5C_1 - 2T_1 \geqq 2(k + \Delta) \geqq 0 \tag{4.27}$$

At X_2,

$$C_3' = 2(T_1 - C_1)$$

$$\text{and} \quad T_3' = 2(T_1 - C_1 + C_2 + k)$$

$$U' = C_1/T_1 + (C_2 + k)/T_2 + (T_1 - C_1)/(T_1 - C_1 + C_2 + k)$$

$$U' - U = k/T_2 + (T_1 - C_1)/(T_1 - C_1 + C_2 + k) - (T_1 - C_1)/\{2(T_1 - C_1) + \Delta\}$$

$$= \frac{k\,\{3T_1\,(T_1 - 2C_1) + C_1(5C_1 - 2T_1)\}}{(T_1 + C_1 + \Delta)(2T_1 - 2C_1 + \Delta + k)(2T_1 - 2C_1 + \Delta)}$$

Using (4.1) and (4.27), we obtain $U' - U \geqq 0$.

Hence, for given C_1, T_1, T_2 if we choose C_2, C_3 and T_3 such that the utilization factor of the three jobs (C_1, T_1), (C_2, T_2) and (C_3, T_3) is less than or equal to

$$U = C_1/T_1 + (T_2 - 2C_1)/T_2 + 2(T_1 - C_1)/2(T_1 - C_1 + T_2),$$

then the three jobs can be feasibly scheduled on two processors according to the rate-monotonic-first-fit scheduling algorithm.

Since $2T_1 \geqq T_2 \geqq T_1 + C_1$, we can write $T_2 = T_1 + C_1 + k$ subject to $0 \leqq k \leqq T_1 - C_1$, and then we can write

$$U = C_1/T_1 + (T_1 - C_1 + k)/(T_1 + C_1 + k) + 2(T_1 - C_1)/(4T_1 - 4C_1 + 2k)$$

subject to the condition that $0 \leqq k \leqq T_1 - C_1$.

Also since the minimum of $f(T)$ occurs for $T \leqq 2T_1 + C_1$, we have the condition that

$$2(2(T_1 - C_1) + k) \leqq 2T_1 + C_1$$

or, $$2T_1 - 5C_1 + 2k \leqq 0.$$

Writing x for C_1/T_1 and y for k/T_1, we can write

$$U = x + (1 - x + y)/(1 + x + y) + (1 - x)/(2 - 2x + y) \quad (4.28)$$

subject to:

$$\text{(i)} \quad 0 \leqq y \leqq 1 - x \quad (4.29)$$

$$\text{(ii)} \quad 2 - 5x + 2y \leqq 0 \quad (4.30)$$

In order to find the minimum U over all possible combinations of C_1, T_1 and T_2 that satisfy the initial conditions of this case, we have to minimize U subject to the conditions (4.29) and (4.30). This can be done by forming the Lagrangian

$$L = U + \lambda(2 - 5x + 2y)$$

and minimizing L.

$$\frac{\partial L}{\partial x} = 1 - \frac{2(1 + y)}{(1 + x + y)^2} - \frac{y}{\{2(1 - x) + y\}^2} - 5\lambda = 0$$

$$\frac{\partial L}{\partial y} = \frac{2x}{(1 + x + y)^2} - \frac{}{\{2(1 - x) + y\}^2} + 2\lambda = 0$$

$$\frac{\partial L}{\partial \lambda} = 2 - 5x + 2y = 0$$

Solving these equations for x, y and λ, we obtain

$$x = \sqrt{6} - 2$$

$$y = 5\sqrt{6}/2 - 6$$

$$\lambda = (3 - \sqrt{6})/3 - 4/49(\sqrt{6} - 2)$$

Then we get $U = 2(\sqrt{6} - 2) + 3/7 > 3/(1 + 2^{1/3})$

Thus in this case also, a set of three jobs with utilization factor less than or equal to $3(1 + 2^{1/3})$ can be feasibly scheduled on two processors according to the rate-monotonic-first-fit scheduling algorithm.

<u>Case 3:</u> $2T_1 \leq T_2 \leq 3T_1$

In this case T_3 must be less than $3T_1$, because otherwise,

$$C_1/T_1 + C_2/T_2 > 2(\sqrt{6} - 2)$$

$$C_1/T_1 + C_3/T_3 > 2(\sqrt{12} - 3)$$

$$C_2/T_2 + C_3/T_3 > 2(\sqrt{2} - 1)$$

which gives $U = C_1/T_1 + C_2/T_2 + C_3/T_3 > 3(1 + 2^{1/3})$, contrary to our initial condition.

Thus we have to consider only the case $T_3 < 3T_1$.

In figure 4.10, the curve AA' represents the function $f_1(T)$ and the curve BB' represents the function $f_2(T)$.

It can be easily shown that the two functions $f_1(T)$ and $f_2(T)$ must have a common point somewhere in the interval $[2T_1 + C_1, 3T_1]$.

Let X_1 be this point of intersection. Then the function $f(T)$ is represented by the curve BX_1A'. Since the function $f_1(T)$ is increasing in the interval $[2T_1 + C_1, 3T_1]$ and $f_2(T)$ is decreasing in this interval, the minimum of $f(T)$ in this interval will be at the point of intersection of the two functions $f_1(T)$ and $f_2(T)$ at X_1.

At the point X_1,

$$T_2 - C_2 = T - 3C_1$$

so that, $T = T_2 - C_2 + 3C_1$ and $C = T_2 - C_2$

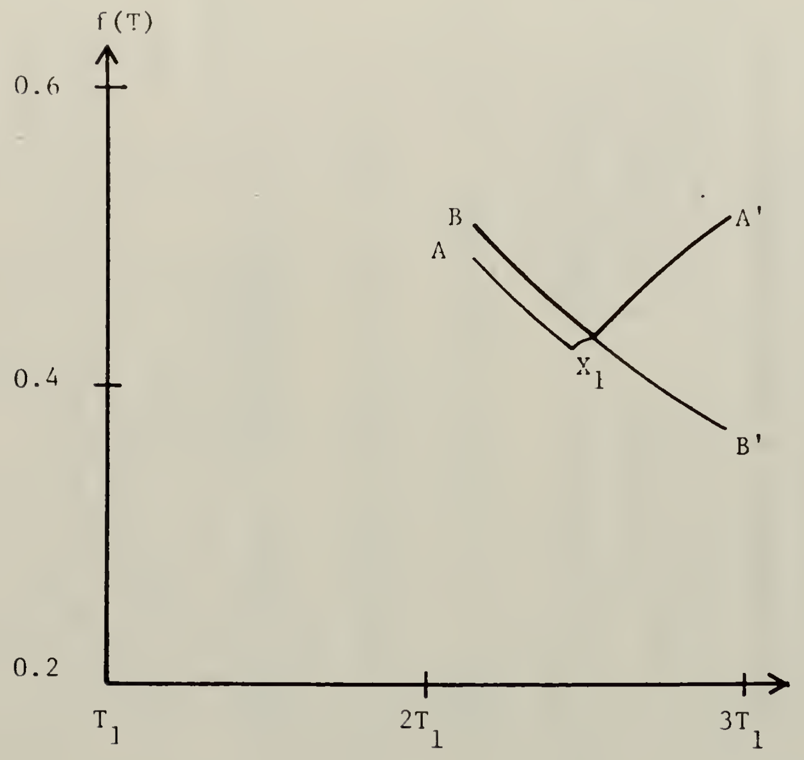

Figure 4.10 Graph representing the maximum utilization factor of a job that fully utilizes processor P_1 along with job J_1, and processor P_2 along with job J_2.

For fixed values of C_1, T_1 and T_2, we can choose C_2 in the range ($2(T_1 - C_1)$, $\frac{1}{2}T_2$), and then choose C_3, T_3 such that f(T) is minimum at T_3. It is claimed that $U = C_1/T_1 + C_2/T_2 + C_3/T_3$ will be minimum when C_2 is minimum possible, and C_3 and T_3 are $T_2 - C_2$ and $T_2 - C_2 + 3C_1$ respectively. In that case,

$$U = C_1/T_1 + C_2/T_2 + (T_2 - C_2)/(T_2 - C_2 + 3C_1)$$

Suppose $C_2' > C_2$.

To be specific, let $C_2' = C_2 + k$, $k > 0$

Then, $C_3' = T_2 - C_2 - k = C_3 - k$

and $T_3' = T_2 - C_2 - k + 3C_1 = T_3 - k$

$$U' = C_1/T_1 + (C_2 + k)/T_2 + (T_2 - C_2 - k)/(T_2 - C_2 - k + 3C_1)$$

$$= C_1/T_1 + (C_2 + k)/T_2 + (C_3 - k)/(T_3 - k)$$

$$U' - U = k/T_2 + (C_3 - k)/(T_3 - k) - C_3/T_3$$

$$= k\,\{T_3(T_3 - k) - T_2\,(T_3 - C_3)\}/T_2T_3(T_3 - k)$$

Since $T_3 \geqq T_2$ and $T_3 - k > T_3 - C_3$

$U' - U > 0$.

Thus U is minimum when C_2 is minimum.

There are two cases to be considered:

(i) $2T_1 \leqq T_2 \leqq 2T_1 + C_1$

and (ii) $2T_1 + C_1 \leqq T_2 \leqq 3T_1$

In case (i), $C_2 > 2(T_1 - C_1)$

Thus, in this case, minimum U is given by

$$U = \frac{C_1}{T_1} + \frac{2(T_1 - C_1)}{T_2} + \frac{T_2 - 2(T_1 - C_1)}{T_2 - 2(T_1 - C_1) + 3C_1}$$

Since $T_2 > 2C_2$, we can write

$$T_2 = 2C_2 + k, \quad \text{where } k > 0$$
$$= 4(T_1 - C_1) + k$$

This gives

$$U = \frac{C_1}{T_1} + \frac{2(T_1 - C_1)}{4(T_1 - C_1) + k} + \frac{2(T_1 - C_1) + k}{2T_1 + C_1 + k}$$

subject to the condition that

$$2T_1 \leqq 4(T_1 - C_1) + k \leqq 2T_1 + C_1$$

or $$0 \leqq 2T_1 - 4C_1 + k \leqq C_1.$$

Writing x for C_1/T_1 and y for k/T_1,

$$U = x + \frac{2(1 - x)}{4(1 - x) + y} + \frac{2(1 - x) + y}{2 + x + y}.$$

subject to the condition that

$$0 \leqq 2 - 4x + y \leqq x$$

In order to minimize U over all x and y, we set

$$\frac{\partial U}{\partial x} = 1 - \frac{2y}{\{4(1 - x) + y\}^2} - \frac{3(2 + y)}{(2 + x + y)^2} = 0$$

$$\frac{\partial U}{\partial y} = - \frac{2(1 - x)}{\{4(1 - x) + y\}^2} - \frac{3x}{(2 + x + y)^2} = 0$$

Solving these two equations for x and y, we obtain

$$x = \frac{\sqrt[3]{18} - \sqrt[3]{12} + 2}{5}$$

$$y = (x^2 + 4x - 2)/(1 - x)$$

and the corresponding value of

$$U = 3x = \frac{3}{5}\{\sqrt[3]{18} - \sqrt[3]{12} + 2\} > 3/(1 + 2^{1/3}).$$

In case (ii),

Let $T_2 = 2T_1 + C_1 + k \leqq 3T_1, \quad k \geqq 0$

Then $\quad C_2 = 2(T_1 - C_1) + k$

Therefore, $U = C_1/T_1 + C_2/T_2 + (T_2 - C_2)/(T_2 - C_2 + 3C_1)$

$$= C_1/T_1 + (2T_1 - 2C_1 + k)/(2T_1 + C_1 + k) + \tfrac{1}{2}$$

subject to the condition that

$$T_2 > 2C_2$$

or $\quad 5C_1 > 2T_1 + k$

Again, writing x for C_1/T_1 and y for k/T_1,

$$U = x + (2 - 2x + y)/(2 + x + y) + \tfrac{1}{2}$$

subject to the condition that

$$5x > 2 + y \tag{4.29}$$

In order to minimize U over all possible pairs x and y, we set

$$\frac{\partial U}{\partial x} = 1 - \frac{3(2 + y)}{(2 + x + y)^2} = 0 \tag{4.30}$$

$$\frac{\partial U}{\partial y} = \frac{3x}{(2 + x + y)^2} = 0 \tag{4.31}$$

(4.31) gives $x = 0$, violating condition (4.29). Hence $x \neq 0$; but then $\partial U/\partial y \neq 0$. Hence y itself must be 0.

Then, $\partial U/\partial x = 1 - 6/(2 + x)^2 = 0$,

giving $\quad x = \sqrt{6} - 2$.

and $\quad U = 2(\sqrt{6} - 2) + \tfrac{1}{2} > 3/(1 + 2^{1/3})$.

Hence, if a set of three jobs cannot be feasibly scheduled on two processors according to the rate-monotonic-first-fit scheduling algorithm, the sum of their utilization factors must be greater than $3/(1 + 2^{1/3})$.

Theorem 6: If a set of m jobs cannot be feasibly scheduled on (m - 1) processors according to the rate-monotonic-first-fit scheduling algorithm, then the utilization factor of the set of jobs must be greater than $m/(1 + 2^{1/3})$.

Proof: Since no set of 3 jobs can be feasibly scheduled on two processors, the utilization factor of any subset of three jobs is greater than $3/(1 + 2^{1/3})$. Hence,

$$
\begin{aligned}
u_1 + u_2 + u_3 &> 3/(1 + 2^{1/3}) \\
u_1 + u_2 + u_4 &> 3/(1 + 2^{1/3}) \\
&\vdots \\
u_1 + u_2 + u_m &> 3/(1 + 2^{1/3}) \\
u_2 + u_3 + u_4 &> 3/(1 + 2^{1/3}) \\
&\vdots \\
u_2 + u_3 + u_m &> 3/(1 + 2^{1/3}) \\
&\vdots \\
u_{m-2} + u_{m-1} + u_m &> 3/(1 + 2^{1/3})
\end{aligned}
$$

where u_i is the utilization factor of job J_i, i = 1, 2, ... , m.

Summing up all these inequalities, we obtain:

$$\binom{m-1}{2}(u_1 + u_2 + \ldots\ldots + u_m) > \binom{m}{3} 3/(1 + 2^{1/3})$$

or

$$\sum_{i=1}^{m} u_i > m/(1 + 2^{1/3}),$$

as claimed.

Theorem 7: Let N be the number of processors required to feasibly schedule a set of jobs by the rate-monotonic-first-fit scheduling algorithm, and N_0 be the minimum number of processors required to feasibly schedule the same set of jobs. Then, as N_0 approaches infinity:

$$2 \leqq \lim_{N_0 \to \infty} N/N_0 \leqq 4 \times 2^{1/3}/(1 + 2^{1/3})$$

Before we give the proof of this theorem, we establish a series of lemmas.

We define first a function f mapping the utilization factors of jobs into the real interval [0, 1]. If u is the utilization factor of a job, let

$$f(u) = \begin{cases} 2u & 0 \leqq u \leqq 1/2 \\ 1 & 1/2 \leqq u \leqq 1 \end{cases}$$

Lemma 1: If jobs are assigned to the processors according to rate-monotonic-first-fit scheduling algorithm, amongst all processors to each of which two jobs are assigned, there is at most one processor for which the utilization factors of the set of the two jobs is less than 1/2.

Proof: Suppose the contrary is true. Let J_{r1} and J_{r2} denote the two jobs assigned to processor P_r, and J_{s1} and J_{s2} denote the two jobs assigned to processor P_s $(r < s)$, such that

$$u_{r1} + u_{r2} < 1/2 \qquad (4.32)$$

and

$$u_{s1} + u_{s2} < 1/2 \qquad (4.33)$$

We have the following three cases:

<u>Case 1</u>: Jobs J_{s1} and J_{s2} were assigned to processor P_s after job J_{r2} had been assigned to processor P_r. Since a set of three jobs with utilization factor less than or equal to $3(2^{1/3} - 1)$ can be feasibly scheduled on a single processor according to the rate-monotonic scheduling algorithm (Theorem A, chapter 2), we must have:

$$u_{r1} + u_{r2} + u_{s1} > 3(2^{1/3} - 1)$$

and $$u_{r1} + u_{r2} + u_{s2} > 3(2^{1/3} - 1)$$

Hence, $$u_{s1} + u_{s2} > 6(2^{1/3} - 1) - 2(u_{r1} + u_{r2})$$

$$> 6(2^{1/3} - 1) - 1$$

or, $$u_{s1} + u_{s2} > 1/2,$$

which is a contradiction to (4.33) above.

<u>Case 2</u>: Jobs J_{s1} and J_{s2} were assigned to processor P_s after job J_{r1} had been assigned to processor P_r, but prior to the assignment of job J_{r2}. We have, in this case,

$$u_{r1} + u_{s1} > 2(2^{1/2} - 1)$$

$$u_{r1} + u_{s2} > 2(2^{1/2} - 1)$$

Hence, $$u_{s1} + u_{s2} > 4(2^{1/2} - 1) - 2u_{r1}$$

$$> 4(2^{1/2} - 1) - 1 > 1/2,$$

which is again a contradiction to (4.33) above.

Case 3: Job J_{s1} was assigned to processor P_s after job J_{r1} had been assigned to processor P_r, and job J_{s2} was assigned to processor P_s after job J_{r2} had been assigned to processor P_r. We have:

$$u_{r1} + u_{s1} > 2(2^{1/2} - 1)$$

and $$u_{r1} + u_{r2} + u_{s2} > 3(2^{1/3} - 1)$$

Once again, we have

$$\begin{aligned} u_{s1} + u_{s2} &> 2(2^{1/2} - 1) + 3(2^{1/3} - 1) - 1/2 - 1/2 \\ &> 1/2, \end{aligned}$$

which is in contradiction to (4.33) above.

Lemma 2: Let N_0 be the minimum number of processors required to schedule the set of jobs $J_1, J_2, \ldots\ldots, J_m$, with utilization factors $u_1, u_2, \ldots\ldots, u_m$, respectively. Then,

$$\sum_{i=1}^{m} f(u_i) \leqq 2N_0$$

Proof: Let $J_{r1}, J_{r2}, \ldots\ldots\ldots, J_{rk_r}$ be the set of jobs assigned to processor P_r. Since

$$\sum_{i=1}^{k_r} u_{ri} \leqq 1$$

we have

$$\sum_{i=1}^{k_r} f(u_{ri}) \leqq 2 \sum_{i=1}^{k_r} u_{ri} \leqq 2$$

Hence,

$$\sum_{i=1}^{m} f(u_i) = \sum_{r=1}^{N_0} \sum_{i=1}^{k_r} f(u_{ri}) \leq 2N_0,$$

as claimed.

We introduce now some definitions.

Let $J_{r1}, J_{r2}, \ldots\ldots, J_{rk_r}$ be k_r jobs assigned to processor P_r, and let $\sum_{i=1}^{k_r} u_{ri} = U_r$. The deficiency δ_r of processor P_r is defined as:

$$\delta_r = \begin{cases} 0 & \text{if } U_r \geq k_r[2^{1/k_r} - 1] \\ 2[1 + U_r/k_r]^{-k_r} - 1, & \text{otherwise} \end{cases}$$

The coraseness α_r of processor P_r is defined to be:

$$\alpha_r = \begin{cases} 0 & \text{for } r = 1 \\ \max\limits_{1 \leq j \leq r-1} (\delta_j), & \text{for } r > 1. \end{cases}$$

Lemma 3: Suppose jobs are assigned to processors according to the rate-monotonic-first-fit scheduling algorithm. If a processor with coarseness α greater than or equal to 1/6 is assigned three or more jobs, then $\Sigma f(u_i) \geq 1$, where u_i runs over all jobs assigned to the processor.

Proof: First we observe that if the coarseness of a processor is α, then the utilization factor of every job on this processor is larger than α. This follows directly from the definition of the coarseness and Theorem 2. Thus the utilization

factor of each of the jobs assigned to this processor is larger than 1/6. If any one of the jobs assigned to the processor has utilization factor larger than or equal to 1/2, the result is immediate. Otherwise,

$$\Sigma f(u_i) \geqq 2 \times 3 \times 1/6 \geqq 1.$$

Lemma 4: Suppose jobs $J_{r1}, J_{r2}, \ldots\ldots, J_{rk_r}$, $k_r > 2$, are assigned to processor P_r, whose coarseness α_r is less than 1/6. If

$$\sum_{i=1}^{k_r} u_{ri} \geqq \ln 2 - \alpha_r$$

then,

$$\sum_{i=1}^{k_r} f(u_{ri}) \geqq 1.$$

Proof: If any one of the jobs has utilization factor larger than or equal to 1/2, the result is immediate. We, therefore, assume that all jobs have utilization factors less than 1/2. We then have:

$$\sum_{i=1}^{k_r} f(u_{ri}) = 2 \sum_{i=1}^{k_r} u_{ri} \geqq 2(\ln 2 - \alpha_r)$$

$$> 2(\ln 2 - 1/6) > 1.$$

Lemma 5: Let processor P_r with coarseness α_r be assigned jobs $J_{r1}, J_{r2}, \ldots\ldots, J_{rk_r}$, with utilization factors $u_{r1}, u_{r2}, \ldots\ldots, u_{rk_r}$, respectively, and let

$$\sum_{i=1}^{k_r} f(u_{ri}) = 1 - \beta, \text{ where } \beta > 0,$$

then:

either (i) $k_r = 1$ and u_{r1} is less than 1/2,

or (ii) $k_r = 2$ and $u_{r1} + u_{r2}$ is less than 1/2,

or (iii) $\sum_{i=1}^{k_r} u_{ri} \leqq \ln 2 - \alpha_r - \beta/2.$

<u>Proof</u>: (i) If $k_r = 1$ and u_{r1} is greater than or equal to 1/2, then $f(u_{r1}) = 1$, which contradicts the fact that $\beta > 0$.

(ii) If $k_r = 2$, and $u_{r1} + u_{r2} \geqq 1/2$, then again we have $f(u_{r1}) + f(u_{r2}) \geqq 1$, which is in contradiction to the fact that $\beta > 0$.

(iii) If neither (i) nor (ii) holds, then $k_r \geqq 3$. By Lemma 4, we have:

$$\sum_{i=1}^{k_r} u_{ri} < \ln 2 - \alpha_r$$

Let
$$\sum_{i=1}^{k_r} u_{ri} = \ln 2 - \alpha_r - \lambda, \text{ where } \lambda > 0.$$

Let us replace jobs $J_{ri} \equiv (C_{ri}, T_{ri})$, $i = 1, 2, 3$, by jobs $J'_{ri} \equiv (C'_{ri}, T_{ri})$, such that $C'_{ri} \geqq C_{ri}$, and

$$\sum_{i=1}^{3} C'_{ri}/T_{ri} = \sum_{i=1}^{3} C_{ri}/T_{ri} + \lambda$$

and $\quad C'_{ri}/T_{ri} < 1/2 \quad$ for $i = 1, 2, 3.$

Since the utilization factor of the set of jobs J'_{r1}, J'_{r2}, and J'_{r3}, and J_{r4}, ..., J_{rk_r} is ln 2, this set can be feasibly scheduled on a single processor (Theorem A, chapter 2). By Lemma 4,

$$\sum_{i=1}^{3} f(u'_{ri}) + \sum_{i=4}^{k_r} f(u_{ri}) \geqq 1$$

$$\sum_{i=1}^{k_r} f(u_{ri}) \geqq 1 - f(\lambda) = 1 - 2\lambda$$

$$1 - \beta \geqq 1 - 2\lambda$$

$$\sum_{i=1}^{k_r} u_{ri} \leqq \ln 2 - \alpha_r - \beta/2.$$

<u>Proof of Theorem</u>: Suppose that for a given set of jobs N processors were used in the rate-monotonic-first-fit scheduling algorithm. Let P_{r_1}, P_{r_2}, ..., P_{r_s} denote the processors for each of which $\Sigma f(u)$, where the summation runs over all jobs assigned to the processor, is strictly less than 1. For convenience, let us relabel these processors as Q_1, Q_2, ..., Q_s. To be specific, for processor Q_j, let

$$\sum_{r=1}^{k_j} f(u_{jr}) = 1 - \beta_j, \quad \text{where } \beta_j > 0.$$

$$\text{for } j = 1, 2, \ldots\ldots, s.$$

Let us divide all these processors into 3 sets:

(1) Processors to each of which only one job is assigned. Suppose there are p of them.

(2) Processors to each of which two jobs are assigned. According to Lemma 1, there is at most one such processor. Let us denote this number by q, where q = 0 or 1.

(3) Processors to each of which more than two jobs are assigned. Suppose that there are r of them. Clearly $p + q + r = s$.

Note that coarseness of each processor in set (3) is less than 1/6 (Lemma 3). Let α_j be the coarseness of processor Q_j. For the r processors in set (3), we have:

$$U_j = \sum_{i=1}^{k_j} u_{ji} \leqq \ln 2 - \alpha_j - \beta_j/2$$

Also

$$\alpha_{j+1} \geqq \delta_j \geqq \ln 2 - U_j, \quad \text{for } j = 1, 2, \ldots\ldots, r - 1.$$

Thus,

$$\alpha_j + \beta_j/2 \leqq \ln 2 - U_j \leqq \alpha_{j+1}$$

$$\text{for } j = 1, 2, \ldots\ldots\ldots, r - 1.$$

Hence,

$$\frac{1}{2} \sum_{i=1}^{r-1} \beta_i \leqq \alpha_r - \alpha_1 < 1/6.$$

Thus, for the first r-1 processors in set (3),

$$f(u) = (r - 1) - \sum_{i=1}^{r-1} \beta_i$$

$$\geq (r - 1) - 1/3 = r - 4/3.$$

For the processors in set (1), since the p tasks do not fit on p-1 processors, by Theorem 6,

$$\sum_{i=1}^{p} u_{i1} \geq p/(1 + 2^{1/3}) \tag{4.34}$$

Also, since f(u) for each of these processors is less than 1, each of these jobs has utilization factor less than 1/2. Hence,

$$\sum_{i=1}^{p} f(u_{i1}) \geq 2p/(1 + 2^{1/3})$$

Also, no job in this set has utilization factor less than 1/3, because a job with utilization factor less than 1/3 can be feasibly scheduled on a single processor together with a job of utilization factor less than 1/2. Hence in any optimal partition of the set of jobs, no more than two of these jobs can be scheduled on a single processor. Therefore, $N_0 > p/2$.

$$\sum_{i=1}^{n} f(u_i) \geq (N - s) + (r - 4/3) + 2p/(1 + 2^{1/3})$$

$$= N - (p + q + r) + (r - 4/3) + 2p/(1 + 2^{1/3})$$

$$= N - p\,\{1 - 2/(1 + 2^{1/3})\} - 4/3 - q$$

Also, from Lemma 2, we have

$$\sum_{i=1}^{n} f(u_i) \leq 2N_0$$

Therefore, $N \leqq 2N_0 + \dfrac{p\,(2^{1/3} - 1)}{(1 + 2^{1/3})} + 4/3 + q$

$$\frac{N}{N_0} \leqq 2 + \frac{\dfrac{p(2^{1/3} - 1)}{(2^{1/3} + 1)}}{N_0} + \frac{4/3 + q}{N_0}$$

$$\leqq 2 + 7/3N_0 + 2\,\frac{2^{1/3} - 1}{2^{1/3} + 1}$$

When N_0 is sufficiently large, we have

$$N/N_0 \leqq 4.2^{1/3}/(1 + 2^{1/3}) + \varepsilon.$$

To eastblish the lower bound, we show that for a given $\varepsilon > 0$, there exists an arbitrarily large set of jobs for which $N/N_0 > 2 - \varepsilon$.

For a given N, let us choose a set of N jobs as follows:

$$\tau_1 = (1,\ 1 + 2^{1/N})$$

$$\tau_2 = (2^{1/N} + \delta,\ 2^{1/N}(1 + 2^{1/N}))$$

$$\tau_3 = (2^{2/N} + \delta,\ 2^{2/N}(1 + 2^{1/N}))$$

$$\vdots$$

$$\tau_N = (2^{(N-1)/N} + \delta,\ 2^{(N-1)/N}(1 + 2^{1/N}))$$

where δ is such that no job has utilization factor greater than 1/2.

This set of jobs when scheduled according to the rate-monotonic first-fit scheduling algorithm, will require N processors, because no two of these jobs can be feasibly scheduled on a single processor

according to the rate-monotonic scheduling algorithm. However, since the utilization factor of each job in this set is less than 1/2, any pair of these jobs can be feasibly scheduled on a single processor according to the deadline driven scheduling algorithm. Thus all these jobs can be feasibly scheduled on $\lceil \frac{N}{2} \rceil$

Thus, $N_0 \leqq \lceil \frac{N}{2} \rceil$ and so, $N/N_0 \geqq N/\lceil \frac{N}{2} \rceil$

Taking N sufficiently large, we can make the ratio $N/N_0 > 2 - \varepsilon$.

CHAPTER 5

CONCLUSIONS

The problem of scheduling periodic-time-critical jobs on single and multiple processor computing systems was considered in the previous chapters. In view of the difficulty in devising optimal algorithms for scheduling periodic-time-critical jobs on multi-processor computing systems, we directed our attention to two heuristic algorithms. These heuristic algorithms do not use more than a fixed percentage of the minimum number of processors that are needed. We have obtained bounds on the worst-case behavior of these algorithms. In the case of the rate-monotonic-next-fit scheduling algorithm, we showed that in the worst case, the ratio N/N_0, where N is the number of processors needed according to the algorithm, and N_0 is the minimum number of processors, is lower bounded by the constant 2.4 and is upper bounded by the constant 2.67. We suspect that the upper bound can be improved to 2.4, although we have not been able to prove it. In the case of the rate-monotonic-first-fit scheduling algorithm, we showed that in the worst case, the ratio N/N_0 is lower bounded by the constant 2 and is upper bounded by the constant $4 \times 2^{1/3}/(1 + 2^{1/3})$. Again, we suspect that the upper bound can be improved to 2. In Theorem 6, we showed that if a set of m jobs, $m \geq 3$, cannot be feasibly scheduled on m - 1 processors according to the rate-monotonic-first-fit scheduling algorithm then the utilization factor of the set of jobs must be greater than $m/(1 + 2^{1/3})$. If for a given

set of jobs, the ratio of the longest request period to the shortest period is less than or equal to 2, we are able to show that this value can be increased to $m/(1 + 2^{1/m})$. It is our conjecture that this result is true for any arbitrary set of m jobs. If this can be proved then substituting $p/(1 + 2^{1/p})$ for $p/(1 + 2^{1/3})$ in equation (4.34), we see that the upper bound for the rate-monotonic-first-fit scheduling algorithm can be reduced to 2. We have:

Theorem 8: If a set of m jobs $\{J_i \equiv (C_i, T_i)\}_{i=1}^{m}$, with $T_1 \leqq T_2 \leqq \ldots\ldots \leqq T_m \leqq 2T_1$, cannot be feasibly scheduled on m-1 processors according to the rate-monotonic-first-fit scheduling algorithm, then the utilization factor of this set of jobs must be greater than $m/(1 + 2^{1/m})$.

Proof: We shall prove the theorem by induction on m. We know that the result is true for m = 2 and m = 3. Let us suppose, it is true for all integers less than m.

We first observe that the utilization factor of each of the jobs in the set is less than 1/2. This follows from the fact that no subset with m-1 jobs can be feasibly scheduled on m-2 processors, or else the given set could be feasibly scheduled on m-1 processors. Hence, the utilization factor of any m-1 jobs in the set is greater than $(m - 1)/(1 + 2^{1/(m-1)})$. Thus if any one of the jobs has utilization factor greater than or equal to 1/2, then the utilization factor of the set will be greater than $(m - 1)/(1 + 2^{1/(m-1)}) + 1/2 > m/(1 + 2^{1/m})$.

Since no two of the jobs in the set can be feasibly scheduled on one processor by the rate-monotonic scheduling algorithm, while assigning jobs to processors according to the rate-monotonic-first-fit

scheduling algorithm, jobs J_1, J_2, ..., J_{m-1}, will be assigned to processors P_1, P_2, ..., P_{m-1} respectively. As T varies, let $f_1(T)$, $f_2(T)$, ... $f_{m-1}(T)$ represent the utilization factor of a job that along with J_1, J_2, ..., J_{m-1} fully utilizes processors P_1, P_2, ..., P_{m-1} respectively. Then, for a given T_m, the maximum utilization factor of the job that can be assigned to at least one of the P_{m-1} processors according to the rate-monotonic-first-fit scheduling algorithm is $\max\{f_1(T_m), f_2(T_m), ..., f_{m-1}(T_m)\}$. Thus, once we fix C_1, T_1, C_2, T_2, ..., C_{m-1}, T_{m-1}, then if the utilization factor of the m^{th} job, (C_m, T_m), is less than or equal to $\min\{\max\{f_1(T), f_2(T), ..., f_{m-1}(T)\}\}$, the m jobs $\{J_i \equiv (C_i, T_i)\}_{i=1}^{m}$, can be feasibly scheduled on m - 1 processors. If we restrict the value of T to within the range T_1 and $2T_1$, then this minimum occurs at the point where the curves $f_1(T)$ and $f_{m-1}(T)$ intersect. This follows from the fact that since $2C_i < T_i$, i = 1, 2, ..., m-1, and $C_{i+1} > T_i - C_i$, for i = 1, 2, ... , m-1, we have $C_{i+1} > C_i$, i = 1, 2, ..., m - 1. Therefore,

$$\max_{T_{m-1} \leqq T \leqq 2T_1} \{f_1(T), ..., f_{m-1}(T)\} = \begin{cases} f_{m-1}(T), & T_{m-1} \leqq T \leqq T' \\ f_1(T), & T' \leqq T \leqq 2T_1 \end{cases}$$

where T' is the point where the two curves $f_1(T)$ and $f_{m-1}(T)$ intersect.

At this point of intersection, we have

$$\frac{T - 2C_1}{T} = \frac{T_{m-1} - C_{m-1}}{T}$$

or

$$T = T_{m-1} - C_{m-1} + 2C_1.$$

Thus, if the utilization factor of the m^{th} job is less than or equal to $(T_{m-1} - C_{m-1})/(T_{m-1} - C_{m-1} + 2C_1)$, then the m jobs can be

feasibly scheduled on m-1 processors. Hence the utilization factor of the given set of jobs must be greater than $[C_1/T_1 + C_2/T_2 + \ldots\ldots + C_{m-1}/T_{m-1} + (T_{m-1} - C_{m-1})/(T_{m-1} - C_{m-1} + 2C_1)]$. We wish to find the minimum value of this expression over all possible combinations of $C_1, T_1, C_2, T_2, \ldots\ldots, C_{m-1}, T_{m-1}$. Since J_1 and J_2 cannot be scheduled on processor P_1, C_2 must be greater than $T_1 - C_1$. Similarly, since J_1 and J_3 cannot be scheduled on P_1, and J_2 and J_3 cannot be scheduled on P_2, we see that $C_3 > \max(T_1 - C_1, T_2 - C_2)$. In general, $C_i > \max(T_1 - C_1, T_2 - C_2, \ldots\ldots, T_{i-1} - C_{i-1})$, $i = 2, 3, \ldots., m$. It can be shown that the expression $[C_1/T_1 + C_2/T_2 + \ldots.. + C_{m-1}/T_{m-1} + (T_{m-1} - C_{m-1})/T_{m-1} - C_{m-1} + 2C_1)]$ will have minimum value when $C_i = T_{i-1} - C_{i-1}$, for $i = 2, 3, \ldots\ldots, m-1$. With these values of C_i's, the utilization factor of the given set of m tasks must be greater than $[C_1/T_1 + (T_1 - C_1)/T_2 + (T_2 - T_1 + C_1)/T_3 + \ldots\ldots\ldots\ldots +$
$(T_{m-2} - T_{m-3} + T_{m-4} - \ldots\ldots\ldots \quad C_1)/T_{m-1} +$
$(T_{m-1} - T_{m-2} + T_{m-3} - \ldots\ldots\ldots \quad C_1)/(T_{m-1} - C_{m-1} + 2C_1)]$.
Since $T_i > 2C_i$, for $i = 2, 3, \ldots\ldots, m-1$, we can write

$$T_2 = 2(T_1 - C_1) + k_2$$

$$T_3 = 2(T_2 - T_1 + C_1) + k_3 = 2(T_1 - C_1 + k_2) + k_3$$

$$\vdots$$

$$T_{m-1} = 2(T_{m-2} - T_{m-3} + T_{m-4} - \ldots\ldots\ldots \quad C_1) + k_{m-1}$$

$$= 2(T_1 - C_1 + k_2 + k_3 + \ldots\ldots + k_{m-2}) + k_{m-1}$$

where k_i's are greater than 0.

Thus we can write the utilization factor as

$$U = C_1/T_1 + (T_1 - C_1)/[2(T_1 - C_1) + k_2] + \ldots\ldots\ldots\ldots\ldots +$$

$$[T_1 - C_1 + k_2 + \ldots + k_{m-2}]/[2(T_1 - C_1 + k_2 + .. + k_{m-2}) + k_{m-1}]$$

$$+ [T_1 - C_1 + k_2 + .. + k_{m-1}]/[T_1 + C_1 + k_2 + \ldots\ldots + k_{m-1}].$$

Writing x_1 for C_1/T_1 and x_i for k_i/T_1, $i = 2, 3, \ldots., m-1$, we have:

$$\begin{aligned} U = x_1 &+ (1 - x_1)/[2(1 - x_1) + x_2] + \ldots\ldots\ldots\ldots + \\ &(1 - x_1 + x_2 + \ldots. + x_{m-2})/[2(1 - x_1 + x_2 + .. + x_{m-2}) + x_{m-1}] \\ &+(1 - x_1 + x_2 + \ldots + x_{m-1})/(1 + x_1 + \ldots.. + x_{m-1}) \end{aligned} \tag{5.1}$$

In order to minimize U over $x_1, x_2, \ldots\ldots, x_{m-1}$, we solve the following set of equations for x_i's:

$$\begin{aligned} \partial U/\partial x_1 &= 1 - x_2/[2(1 - x_1) + x_2]^2 - x_3/[2(1 - x_1 + x_2) + x_3]^2 - \ldots. \\ &- 2(1 + x_2 + \ldots.. + x_{m-1})/[1 + x_1 + \ldots + x_{m-1}]^2 = 0 \end{aligned}$$

$$\begin{aligned} \partial U/\partial x_2 &= -(1 - x_1)/[2(1 - x_1) + x_2]^2 + x_3/[2(1 - x_1 + x_2) + x_3]^2 + .. \\ &+ 2x_1/[1 + x_1 + \ldots + x_{m-1}]^2 = 0 \end{aligned}$$

.
.
.

$$\begin{aligned} \partial U/\partial x_{m-1} &= - x_{m-2}/[2(1 - x_1 + x_2 + .. + x_{m-2}) + x_{m-1}]^2 \\ &+ 2x_1/[1 + x_1 + \ldots\ldots + x_{m-1}]^2 = 0 \end{aligned}$$

On substituting the values of x_i's obtained by solving this set of equations, in (5.1), we obtain:

$$U = m/(1 + 2^{1/m}).$$

This proves our assertion.

There are still many questions in connection with the scheduling of periodic-time-critical jobs. For example, it would be interesting to investigate other heuristic algorithms, such as, the rate-monotonic-best-fit algorithm. It would also be interesting to study the scheduling problem when the processors are not identical. Furthermore, we limited our study to preemptive scheduling algorithms only. The problem of non-preemptive scheduling still remains mostly unexplored. It may also be interesting to study algorithms that minimize the number of preemptions in a schedule.

LIST OF REFERENCES

[1] Chen, N. F. and C. L. Liu, "Bounds on the Critical Path Scheduling Algorithm for Multiprocessors Computing Systems", (to appear).

[2] Coffman, Jr., E.G. and R. L. Graham, "Optimal Scheduling for Two Processor Systems", Acta Informatica 1,3 (1972), pp. 200-213.

[3] Cook, S. A., "The Complexity of Theorem Proving Procedures", Proceedings of the 3rd ACM Symposium on Theory of Computing, 1970, pp. 151-158.

[4] Fujii, M., T. Kasami, and K. Ninomiya, "Optimal Sequencing of Two Equivalent Processors", SIAM J. Appl. Math. 17, 4 (July, 1969), pp. 784-789; Erratum 20, 1 (January 1971), p.141.

[5] Garey, M. R., and R. L. Graham, "Bounds for Multiprocessor Scheduling with Resource Constraints", SIAM J. on Computing, 4 (1975), pp. 187-200.

[6] Graham, R. L., "Bounds on Multiprocessing Timing Anomalies", SIAM J. Appl. Math. 17,2 (March 1969), pp. 416-429.

[7] Hu, T. C., "Parallel Scheduling and Assembly Line Problems", Oper. Res. 9,6 (November 1961), pp. 841-848.

[8] Johnson, D. S., A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-Case Performance Bounds for Simple One Dimensional Packing Algorithms", SIAM J. on Computing, 3 (1974), pp. 299-325.

[9] Karp, R. M., "Reducibility Among Combinatorial Problems", Complexity of Computer Computations, R. E. Miller and J. W. Thatcher (eds.) Plenum Press, New York, N.Y. (1972), pp. 85-104.

[10] Labetoulle, J., "Ordonnancement des processus temps reel sur une ressource préemptive", Thess de 3eme cycle, Universite Paris VI (1974).

[11] Labetoulle, J., "Real Time Scheduling in a Multiprocessor Environment", (to appear).

[12] Lam, S., and R. Sethi, "Worst Case Analysis of Two Scheduling Algorithms", to appear in SIAM J. on Computing.

[13] Liu, C. L., and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", J. ACM, 20 (1973), pp. 46-61.

[14] Liu, J. W. S., and C. L. Liu, "Bounds on Scheduling Algorithms for Heterogeneous Computer Systems", Proceedings of the IFIPS 1974 Congress, North-Holland Publishing Co., August 1974, pp. 349-353.

[15] Sahni, S., "Algorithms for Scheduling Independent Tasks", J. ACM, 23 (1976), pp. 116-127.

[16] Serlin, O., "Scheduling of Time Critical Processes", Proc. of the Spring Joint Computers Conference (1972), pp. 925-932.

[17] Ullman, J. D., "Polynomial Complete Scheduling Problems", 4th Symposium on Operating Systems Principles, Yorktown Heights, New York, (October 1973), pp. 96-101; to appear JCSS

VITA

Sudarshan Kumar Dhall was born in Maghiana, Punjab (now in Pakistan), on September 6, 1937. He received his B.A. degree from the Panjab University, India, in 1956 and M.A. in Mathematics from the University of Delhi, India, in 1968. In 1972, he received M.S. in Mathematics from the University of Illinois at Urbana-Champaign.

He served in the Government of India from October 1955 to January 1970. He was a Research Assistant at the University of Illinois at Urbana-Champaign for six years. He is a member of the American Mathematical Society.

BLIOGRAPHIC DATA EET	1. Report No. UIUCDCS-R-77-859	2.	3. Recipient's Accession No.

Title and Subtitle

Scheduling Periodic-Time-Critical Jobs On Single Processor and Multiprocessor Computing Systems

5. Report Date

6.

Author(s)

Sudarshan Kumar Dhall

8. Performing Organization Rept. No.

Performing Organization Name and Address

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.

MCS-73-03408

Sponsoring Organization Name and Address

National Science Foundation
Washington, DC

13. Type of Report & Period Covered

14.

Supplementary Notes

Abstracts

The problem of presentive scheduling of periodic-time-critical jobs on single processor and multiprocessor computing systems was studied. For the case of multiprocessor computing systems, suboptimal algorithms for assigning jobs to processors were designed and analyzed.

Key Words and Document Analysis. 17a. Descriptors

scheduling theory, multiprocessor computing systems, periodic jobs.

Identifiers/Open-Ended Terms

COSATI Field/Group

vailability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
	20. Security Class (This Page UNCLASSIFIED	22. Price